EM61000 Series V3.1

Integrated Development Environment

USER'S GUIDE

ELAN MICROELECTRONICS CORP.

Fourth Edition
September 2004

Trademark Acknowledgments

IBM is a registered trademark and PS/2 is a trademark of IBM.
Microsoft, MS, MS-DOS, and Windows are registered trademarks of Microsoft Corporation.
Easy Sound® is a registered trademark of ELAN Microelectronics Corp.

© 2000-2004 ELAN Microelectronics Corporation

All Rights Reserved

Printed in Taiwan, ROC, 09/2004

The contents of in this manual is subject to change without notice. ELAN Microelectronics assumes no responsibility for errors that may appear in this manual. ELAN Microelectronics makes no commitment to update, or to keep current, the information contained in this manual. The software described in this manual is furnished under a license or nondisclosure agreement, and may be used or copied only in accordance with the terms of the agreement.

ELAN Microelectronics products are not intended for use in life support appliances, devices, or systems. Use of ELAN Microelectronics products in such applications are not supported and is prohibited.

NO PART OF THE EASY SOUND® AND OF THIS MANUAL MAY BE REPRODUCED OR TRANSMITTED IN ANY FORM OR BY ANY MEANS WITHOUT THE EXPRESS WRITTEN PERMISSION OF ELAN MICROELECTRONICS.

Elan Microelectronics Shanghai, Ltd. ◦

ELAN MICROELECTRONICS CORPORATION

Headquarters:

No. 12, Innovation Road 1
Science-based Industrial Park
Hsinchu, Taiwan, R.O.C. 30077
Tel: +886 3 563-9977
Fax: +886 3 563-9966
http://www.emc.com.tw

Hong Kong Office:

Elan (HK) Microelectronics Corporation, Ltd.
Rm. 1005B, 10/F Empire Centre
68 Mody Road, Tsimshatsui
Kowloon , HONG KONG
Tel: +852 2723-3376
Fax: +852 2723-7780
http://www.elanhk.com.hk

USA Office:

Elan Information Technology Group
10062 Miller Ave., Suite 100
Cupertino, CA 95014
USA
Tel: +1 048 366-8223
Fax: +1 048 366-8220

Europe:

Elan Microelectronics Corp. (Europe)
Dubendorfstrasse 4
8051 Zurich, SWITZERLAND
Tel: +41 43 299-4060
Fax: +41 43 299-4079
http://www.elan-europe.com

Shenzhen Office:

Elan (Shenzhen) Microelectronics Corporation, Ltd.
Rm. 420/4Fl., Fuxing Garden
Fuxing Rd, Futian Dist., Shenzhen
Guandong, CHINA 518031
Tel: +86 755 375-0915/6
Fax: +86 755 375-0911

Shanghai Office:

Elan Microelectronics Shanghai, Ltd.
23/Bldg. 115, Lane 572, Bibo Road
Zhangliang Hi-Tech Park
Shanghai, CHINA
Tel: +86 021 5080-3866
Fax: +86 021 5080-4600

Contents

1 Introduction 1

1.1 Overview 1
1.2 Introduction to EM61IDE Program 2
1.2.1 Tools under File View and Voice View Modes 2
1.2.2 EM61IDE Main User Interface 3
1.2.2.1 The Child Windows 3
1.2.2.2 Menu Bar and its Commands 7
1.2.2.3 Toolbar 10
1.2.2.4 Document Bar 11
1.2.2.5 Status Bar 11
1.3 Introduction to EM61000 ICE 12
1.3.1 EM61000 ICE Top Board 13
1.3.1.1 Top Board JP3 Connector Pin Assignment 14
1.3.2 EM61000 ICE Bottom Board 14

2 System Installation 17

2.1 System Requirements 17
2.1.1 Host Computer 17
2.1.2 External Power Source 17
2.1.3 Audio Speaker 18
2.2 System Installation 18
2.2.1 Connecting EM61000 ICE with Host Computer 18
2.2.1.1 EM61000 ICE Setup 19
2.2.3 Installation of EM61IDE Program to Host Computer 20
2.3 Starting the EM61IDE Program 20

3 Getting Started in File View Mode 21

3.1 Overview 21
3.2 Create a New Project 21
3.2.1 Changing the Target Microcontroller Halfway into the Project 22
3.3 Add and Remove Source Files from/to Project 23
3.3.1 Create New Source File for the Project 23
3.3.2 Add Existing Source Files to the New Project 24
3.3.3 Open Source File or Display its Path 24
3.3.4 Deleting Source Files from Project 25
3.4 Edit a Source File From and Within Project 25
3.5 Assemble and Link the Project 26
3.5.1 Assemble Files to Generate Object File 26
3.5.2 Assemble and Link Files to Generate Binary File 26
3.6 Debugging a Project 27
3.6.1 Defining Breakpoints and Program Counter 28
3.6.1.1 Breakpoint Setup 28
3.6.1.2 Breakpoint Loop Counter Setup 29
3.6.1.3 Enable/Disable Breakpoint from ROM Dialog 29
3.6.2 Debugging Tools 30
3.6.2.1 View Trace Buffer 31
3.7 Generate EPROM and Mask Files from Project 32
3.8 Save, Open, and Close Projects 32

4 Getting Started in Voice View Mode 33

4.1 Overview 33
4.2 Creating an Entirely New Voice Project 33
4.3 Compiling Project 33
4.3.1 Building Voice Data 34
4.3.1.1 Melody Code Header 34
4.3.1.2 How EM61IDE Translate Your MIDI File 35
4.3.2 How to Insert MIDI files into MIDI Node 36
4.3.3 How to Insert User Data into User Data Node 36
4.3.4 How to Insert Instrument Files into Tone Node Manually 37
4.3.5 How to Insert Instrument Files into Percussion Node Manually 38

5 Assembler and Linker 41

5.1 Introduction....41
5.2 Assembler and Linker Process Flow....42
5.3 Assembler Conventions....43
5.3.1 Statement Syntax....43
5.3.2 Numeric Presentation....43
5.3.3 Arithmetic Operations....44
5.3.3.1 True and False....44
5.3.3.2 Summary of the Operators (In the Order of Priority)....44
5.4 Program Directives....45
5.5 Conditional Assembly....49
5.6 Reserved Words....51
5.6.1 Directives, Operators....51
5.6.2 Instruction Mnemonics....51
5.6.3 Register Name....51
5.7 Assembler/Linker Error Messages....52
5.7.1 Assembler Error Messages....52
5.7.2 Linker Error Messages....59

6 Easy Program 61

6.1 Writing an EASY Format® (ESY) Program....61
6.1.1 File Format....61
6.1.1.1 [InputState]:....62
6.1.1.2 [OutputState]:....62
6.1.1.3 [Path]:....63
6.1.1.4 [Background]:....63
6.1.1.5 [Asm]: In-EASY Assembly....64
6.1.1.6 [TCCAsm]:....64
6.1.1.7 [TCC2Asm]....65
6.1.1.8 [Macro]:....65
6.1.1.9 [Symbol]:....65
6.2 Applicable Registers....66
6.3 Path Equation Parameters and Definitions....66
6.4 Reserved Words....74
6.5 Limitations....75
6.6 Guidelines to Writing Assembly Code....75

6.7 Initial Condition 77
6.8 Options for Configuring EASY Project 77
6.8.1 How to Write a Custom Scankey Routine 81
6.8.1.1 “USER_SCAN_KEY_INI:” Label 81
6.8.1.2 “USER_SCAN_KEY:” Label 81
6.8.1.3 “USER_SCAN_KEY_END:” Label 81
6.9 Application Circuits 82
6.9.1 EMV60KB Jumpers & Keyboard Keys Arrangement 82
6.9.2 Direct-4 Application Circuit 82
6.9.2.1 EMV60KB Jumper Configuration for Direct-4 83
6.9.2.2 Applicable EMV60KB Trigger Keys for Direct-4 83
6.9.3 Direct-8 Application Circuit 84
6.9.3.1 EMV60KB Jumper Configuration for Direct-8 84
6.9.3.2 Applicable EMV60KB Trigger Keys for Direct-8 85
6.9.4 Direct-9…16 Application Circuit 85
6.9.4.1 EMV60KB Jumper Configuration for Direct-9…16 86
6.9.4.2 First EMV60KB Applicable Trigger Keys 1…8 86
6.9.4.3 Second EMV60KB Applicable Trigger Keys 9…16 87
6.9.5 EASY-16 Application Circuit 87
6.9.5.1 EMV60KB Jumper Configuration for EASY-16 88
6.9.5.2 Applicable EMV60KB Trigger Keys for EASY-16 88
6.9.6 EASY-24 Application Circuit 89
6.9.6.1 EMV60KB Jumper Configuration for EASY-24 89
6.9.6.2 Applicable EMV60KB Trigger Keys for EAY-24 90
6.9.7 EASY-32 Application Circuit 91
6.9.7.1 EMV60KB Jumper Configuration for EASY-32 91
6.9.7.2 Applicable EMV60KB Trigger Keys for EAY-32 92
6.9.8 EASY-40 Application Circuit 93
6.9.8.1 EMV60KB Jumper Configuration for EASY-40 93
6.9.8.2 Applicable EMV60KB Trigger Keys for EASY-40 94
6.9.9 EASY-48 Application Circuit 94
6.9.9.1 EMV60KB Jumper Configuration for EASY-48 95
6.9.9.2 Applicable EMV60KB Trigger Keys for EASY-48 95
6.9.10 EASY-56 Application Circuit 96
6.9.10.1 EMV60KB Jumper Configuration for EASY-56 96
6.9.10.2 Applicable EMV60KB Trigger Keys for EASY-56 97

6.9.11 EASY-64 Application Circuit....97
6.9.11.1 EMV60KB Jumper Configuration for EASY-64....98
6.9.11.2 Applicable EMV60KB Trigger Keys for EASY-64....98

Appendix

A Voice ROM Data Allocation 99

A.1 Introduction....99
A.2 EM61001/002/100 Voice ROM Configuration....100
A.3 EM61200/300/400/500 Voice ROM Configuration....101
A.4 Data Table Address for Include File....102
A.4.1 For EM61001/002/100....102
A.4.2 For EM61200/300/400/500....103

B MIDI (EM61.mid) File 105

B.1 Compatible MIDI File Format....105
B.2 MIDI Duration....105
B.2.1 Acceptable Minimum Beat....105
B.2.2 Quantization and Resolution....106
B.2.3 Timebase....106
B.3 Frequency....106
B.3.1 Note....106
B.3.2 Base Octave....107
B.4 Patch and Channel in EM61000....107
B.4.1 EM61000 Capabilities....107
B.4.2 Maximum Tones in Channel and Looping....108
B.4.3 Patch Change....108
B.4.3 General MIDI vs. EM61000 Patches....109
B.5 Volume....110
B.5.1 EM61000 Playing Volume....110
B.5.2 Volume Range....110

B.6 Tempo 110
B.6.1 Tempo Range 110
B.6.2 Tempo Limitation 110
B.6.3 Multiple Tempos 111
B.7 Meter 111
B.7.1 Limitations 111
B.8 Controller 111

C Instrument Map (EM61.ist) File 113

C.1 Introduction 113
C.2 [General] Section 116
C.2.1 [General] Example 116
C.3 [Instrument List] Section 116
C.3.1 Tone/Percussion Instrument Group Field Contents 116
C.3.1.1 Tone/Percussion Group Example 117
C.3.2 Tone Instrument Pitch Scale Field Contents 117
C.3.2.1 Pitch Scale Format 118
C.3.2.2 Pitch Scale Example 119
C.3.3 Percussion Instrument Field Contents 121
C.3.3.1 Percussion Instrument Example 121
C.4 [Template List] Section 121
C.4.1 Field Contents 121
C.4.2 Template Waveform Variable format 122
C.4.2.1 Template Waveform Example 122

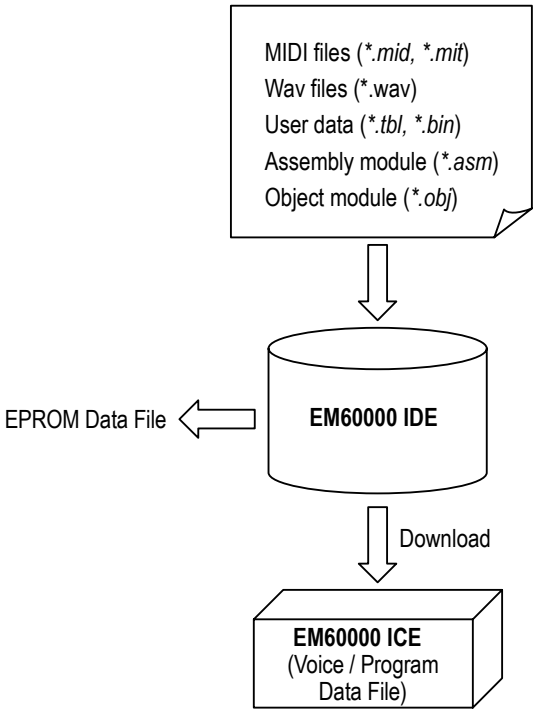
D Envelope Data (EM61.env) File 125

D.1 Introduction 125
D.2 Envelop vs. Play MIDI 126
D.2.1 Summary 126
D.4 Envelope Generator and Data Format 128
D.4.1 Data Format 128
D.5 Illustrative Examples and Demonstrations 130
D.5.1 Common Envelope 130
D.5.2 AD Sustained Without Vibration Envelope 131
D.5.3 AD Sustained with Vibration Envelope 133

Chapter 1
Introduction

1.1 Overview

EM60000 Series **I**ntegrated **D**evelopment **E**nvironment (EM60IDE) is a project oriented development tool for Elan's EM60000 series microcontrollers. It is a fully integrated development environment comprising of the EM60000 ICE (in-circuit emulator) and EM60000 IDE software tool



EM60000 Integrated Development Environment

1.2 Introduction to EM60IDE Program

The EM60IDE Program is a Windows based program that is used among other things, to develop embedded programs for EM60000 series microcontrollers. The embedded program is developed by taking one or more source files in assembly language and translate them into object files that are linked to each other along with one or more libraries. The created executable program is then downloaded to EM60000 ICE for simulation, verification and debugging.

1.2.1 Tools under File View and Voice View Modes

The EM60IDE Program is integrated with the following resourceful tools**:**

File View Mode

- **Project Manager** that create, save, open, and closes project. It also manipulates the existing project and source files that are part of the project
- **Editor** that provides editing functions for creating, viewing, and modifying source files. It supports find, replace, cut/copy/paste and multi-level undo/redo.
- **Assembler** that supports file-include, macro, assemble arithmetic, block comment, conditional assemble, list, and map file.
- **Linker** that supports multiple source and object files linking.
- **Source Level Debugger** that provides source-level debugging function on target embedded on the EM60000 ICE (In Circuit Emulator). You can explore and analyze the status, register, and the memory contents of the EM60000 target with the EM60000 IDE. With its powerful features, like multiple breakpoints, real-time modification of register contents and disassembly, the EM60000 ICE becomes the indispensable pair of EM60000 IDE in offering a perfect development environment for EM60000 series microcontrollers.

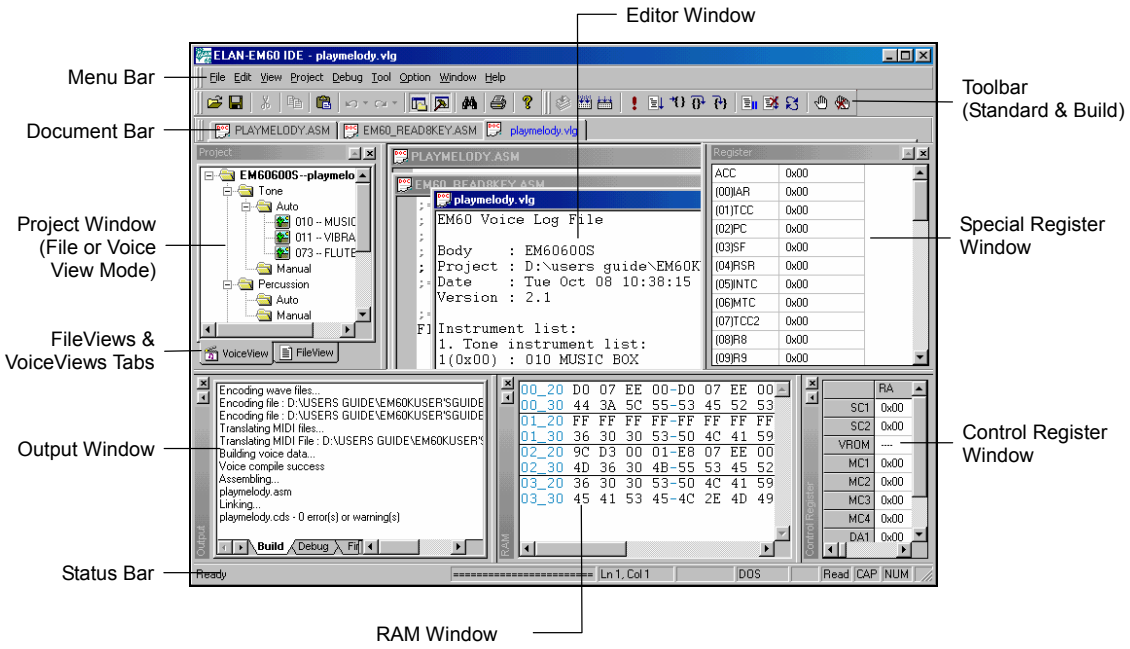
See Chapter 3 for the full discussions of File View Mode functions.

Voice View Mode

- **Project Manager** that create, save, open, and closes project. It also manages and manipulates the existing MIDI files, speech files and user data files to support instrument playback function of the EM60000 series sound processors.
- **Converter** that converts MIDI files , speech files and other text/binary data into voice data files for downloading into your applications.

Voice View Mode functions are discussed separately in Chapter 4.

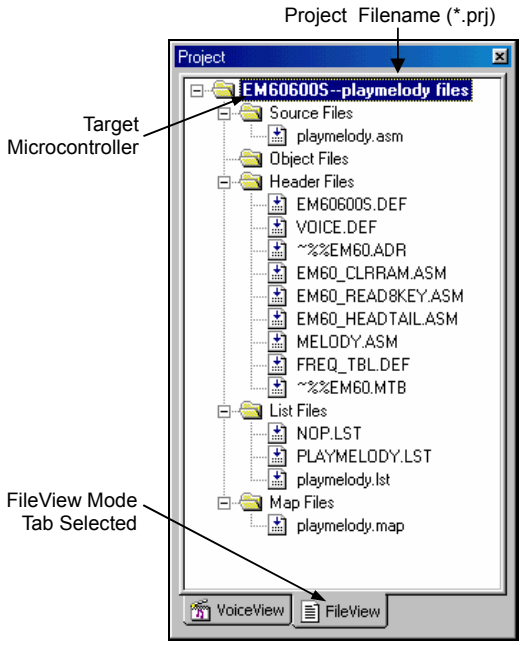
1.2.2 EM60IDE Main User Interface



EM60IDE Main Window Layout

1.2.2.1 The Child Windows

Project Window (File View Mode)



The **Project** window under **FileView** Mode tab holds the Source, Header, List, and Map Files.

Where:

Source Files (**.asm*) – are the source assembly files that are added into the current project. Also located under this node are the Object Files *(*.obj*), which contains the source linked object files.

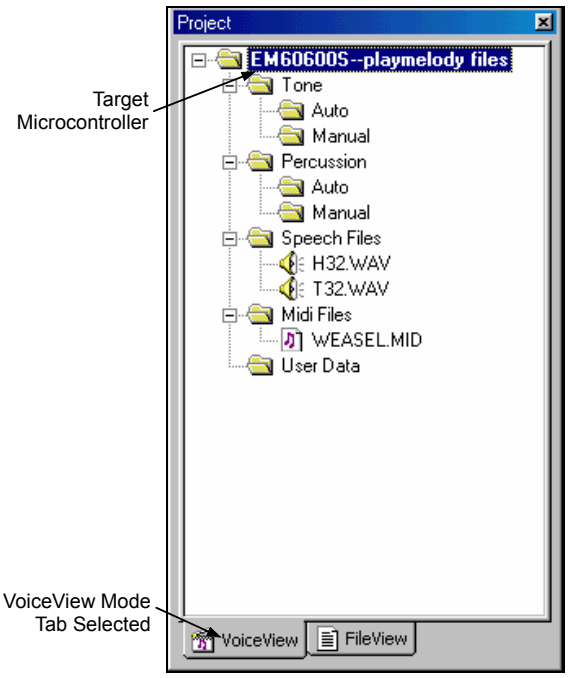
Header Files – are the reference files required by assembly programs.

List Files *(*.lst*) – are the list files of source assembly files.

Map File *(*.map*) – is the file generated by the linker.

The root node of the **Project** window shows your current microcontroller and project filename.

Project Window (Voice View Mode)



The **Project** window under **VoiceView** Mode tab provides the **Tone**, **Percussion**, **Speech Files**, **Midi Files**, and **User Data** nodes that contain files for user's utilization in developing voice projects, where:

Tone Node – stores the musical instruments defined by yoInitially, "Auto" node is empty even if the MIDI files has been added into the voice project. ur MIDI files. The Tone node contains two child nodes, "Auto" and "Manual." Only after the **Build** or **Rebuild All** command is executed from the Project menu (see Chapter 4) that the EM60IDE starts to automatically collect the instruments from MIDI file and display them under "Auto" node. On the other hand, you can manually add instruments into the voice project by clicking the **Insert Tones into Projects**... command from Project menu and select the desired instruments from the resulting "Insert Tone" dialog. The selected instruments are then shown under the "Manual" node.

Percussion Node – contains the percussion instruments defined by your MIDI files. Like the Tone node above, the Percussion node consisted of two child nodes, "Auto" and "Manual." It is also initially empty regardless of the presence of MIDI files in the voice project. Only after clicking the **Build** or **Rebuild All** command from the Project menu (see Chapter 4, Section 4.3.1) that the EM60IDE starts to automatically collect the percussion instruments for MIDI files and display them under "Auto" node. Likewise, you can manually insert percussion instruments into the voice project by executing the **Insert Percussion into Projects**... command from Project menu and select the desired instrument from its "Insert Percussion" dialog. The selected instruments are then displayed under "Manual" node.

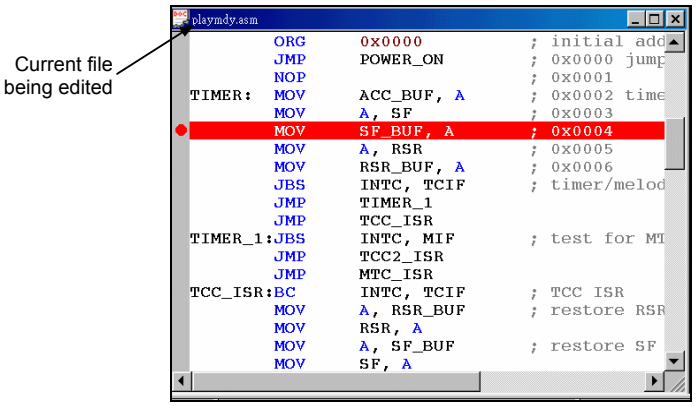
Speech Files – hold and display the wave files of your choice. This is done by clicking the **Insert Files to Projects**... command from the **Project** menu (see Chapter 4). From the resulting "Insert File into Project" dialog, browse for "Speech File (*.wav)" file folder and select the one you want to insert in the voice project. The selected Speech File will show under the "Speech Files" node.

Midi Files – hold and display the MIDI files of your choice. This is done by clicking the **Insert Files to Projects**... command from the **Project** menu (see Chapter 4). From the resulting "Insert File into Project" dialog, browse for "Midi File *(*.mid)*" file folder and select the one you want to insert in the voice project. The selected MIDI file will show under the "MIDI files "node.

> **NOTE**
> *The EM60IDE can only recognize the General MIDI Format 1 of the MMA specifications. Formats 0 and 2 are not supported.*

User Data – contains the data files *(*.tbl* and **.bin)* required by your application. To insert data files, click the **Insert Files to Project**... command from the Project menu. From the resulting **Insert File into Project** dialog, browse for the text format "Text Data File *(*.tbl)*" folder and select the one you want to insert in the voice project. If inserting binary data, browse for the "Binary Data Files (**.bin*)" folder from the same dialog. Press the **OK** button in the dialog to apply selection. The selected data files will be shown under the **User Data** node.

Editor Window


Current file being edited
playmdy.asm
ORG 0x0000 ; initial add
JMP POWER_ON ; 0x0000 jump
NOP ; 0x0001
TIMER: MOV ACC_BUF, A ; 0x0002 time
MOV A, SF ; 0x0003
MOV SF_BUF, A ; 0x0004
MOV A, RSR ; 0x0005
MOV RSR_BUF, A ; 0x0006
JBS INTC, TCIF ; timer/melod
JMP TIMER_1
JMP TCC_ISR
TIMER_1:JBS INTC, MIF ; test for MI
JMP TCC2_ISR
JMP MTC_ISR
TCC_ISR:BC INTC, TCIF ; TCC ISR
MOV A, RSR_BUF ; restore RSR
MOV RSR, A
MOV A, SF_BUF ; restore SF
MOV SF, A

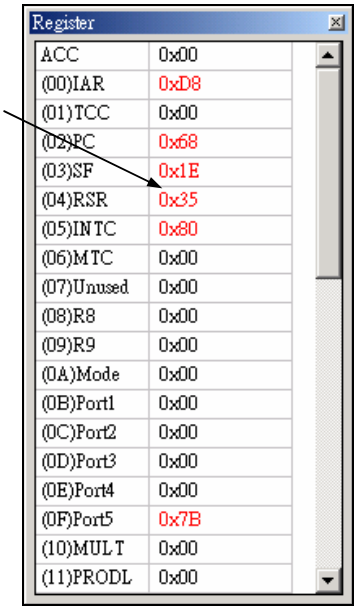
The EM60IDE Editor is a multi-windowed editing tool for creating, viewing, and modifying source files.

The EM60IDE Editor major features are –

- Colored syntax highlighting
- Unlimited file size
- Multiple files can be opened and displayed at the same time
- Insert and overstrike modes for editing
- Insert and overstrike modes for editing
- Unlimited Undo/Redo
- Auto indentation
- Clipboard support with find and replace
- Drag and drop text manipulation

Special Register Window

When value changes, it is shown in red



The **Special Register** window provides the accumulator, control register, a program counter, and general register.

To update register values, click on the existing value and key in the new value.

NOTE

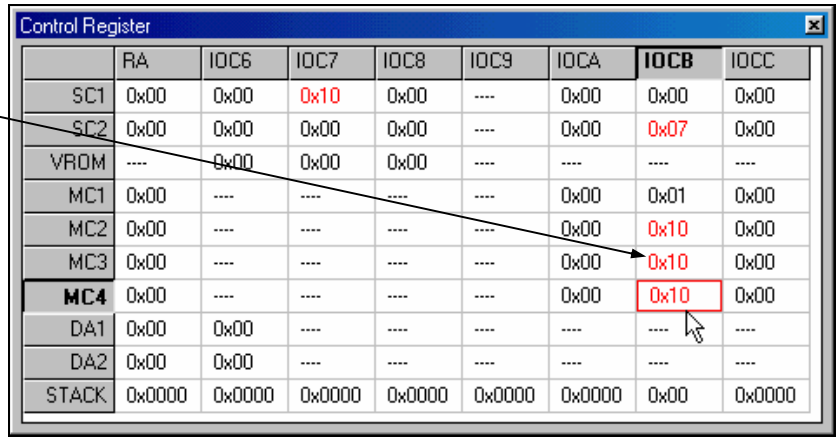
Only values from 0 to 9 or A to F are recognized when changing register in hexadecimal format. Likewise, only 0 & 1 are recognized when changing registers in binary format.

Modified registers are automatically downloaded to the hardware.

To toggle between hex (0x) and binary (0b) register values, click right mouse button while pointing on the particular register value.

Control Register Window

When value changes, it is shown in red



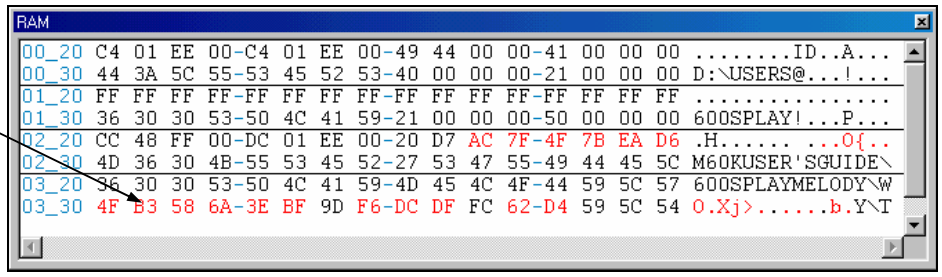
The **Control Register** window incorporates the content of the I/O mapped control registers comprising of the following groups**:**

- SC1 & SC2 – show the banked control registers for speech Channels 1 and 2.
- VROM – displays the Voice ROM access control registers.
- MC1 to MC4 – show the banked control registers for melody Channels 1 to 4.
- DA1 and DA2 – display the banked control registers for D/A Converter 1 and 2.
- STACK – shows the stack register at 2-byte data length. The bottom of the stacked registers is located at the left end column and the stack depth is 8.

You can modify the register content by clicking on the value and key in the desired value.

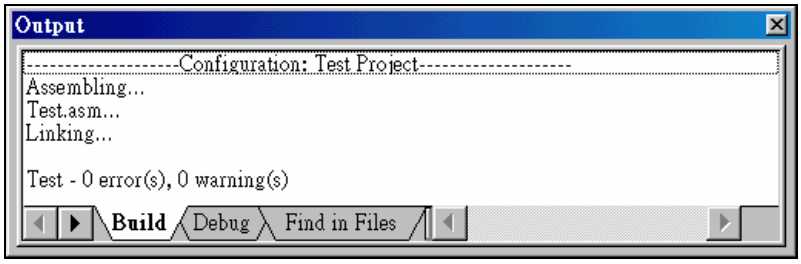
RAM Window

When values are changed, they are shown in red



RAM window provides the updated contents of RAM in the same way as in updating the contents of the special registers. The address format (hex) is XX__DD (example**:** 00_20 means Bank 0 and offset 20).

Output Window



The **Output** window displays messages announcing results of project compiling, such as assembler, linker, trace log history, and debugging, including their respective error messages. The window consisted of three tab sub-windows, namely**; Build**, **Debug**, and **Find in Files**, where**:**

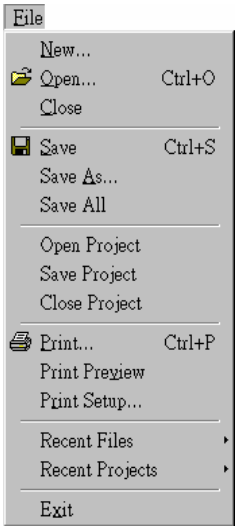
Build – displays assembler/linker related messages, trace logs, and error messages. Double click on the error message to link to the corresponding program text line in the Editor window where the source of error occurs.

Debug – displays debugging related messages and their corresponding error messages.

Find in Files – allows you to find a like string (selected from an active file) from other active or inactive files in your folder. Lines containing the like string will display on the Output window complete with its source filename and directory.

1.2.2.2 Menu Bar and its Commands

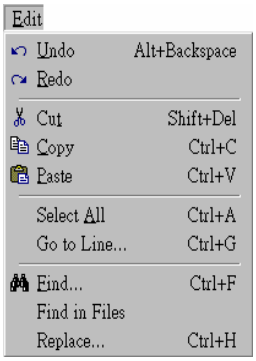
File Menu



New…	Create a new project or source file
Open…	Open an existing document or project
Close	Close the active document or project
Save/Save All	Save current/all active document(s)
Save as	Save the active document with new filename
Open/Save/Close Project	Open/Save/Close the active project
Print/Preview/Setup	Print/preview printed format of the active document/define printer settings
Recent File/Recent Projects	View record of the recently closed source files/projects
Exit	Exit from EM60IDE Program

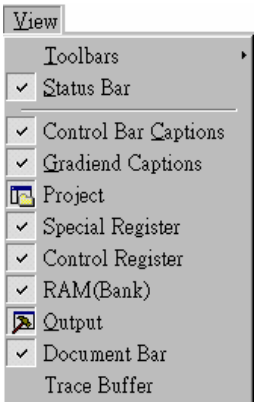
See Chapter 3 for details on how to create a new project as well as manipulate existing ones.

Edit Menu



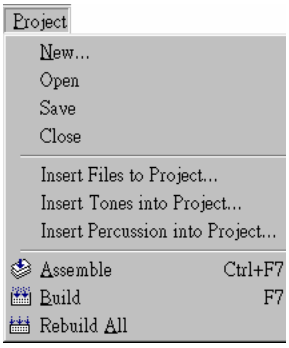
Undo/Redo	Undo last edit action/Cancel last undo
Cut/Copy/Paste	Same as standard clipboard functions
Select All	Select all contents of the active window
Go to Line…	Move cursor to the defined line number within the active window
Find	Find strings in the active window
Find in Files	Find strings within one or more files
Replace…	Same as standard “find & replace” editing functions

View Menu



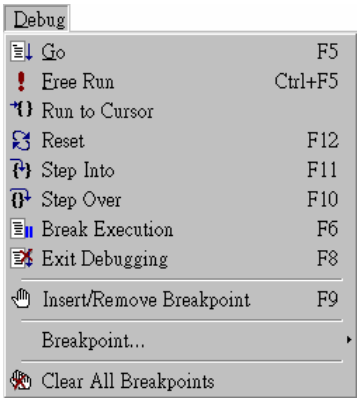
Toolbars	View/hide Standard or Build toolbar
Status Bar	View/hide ICE modes, R/W flag, text file type (DOS/MAC/UNIX), keyboard status, etc.
Control Bar Captions	View/hide all captions for Project, RAM, and Output windows
Gradient Captions	View/hide gradient shading of all mini-windows caption bars
Project	View/hide Project window
Special Register	View/hide Special Register window
Control Register	View/hide Control Register window
RAM (Bank)	View/hide RAM window
Output	View/hide Output window
Document Bar	View/hide Document Bar
Trace Buffer	View/hide Trace Buffer

Project Menu



New…	Create a new project
Open	Open an existing project
Save/Close	Saves the active project together with all related data/Close the active Project window
Insert Files to Project	Insert program/voice source files into project
Insert Tones to Project	Insert music tones into project
Insert Percussion into Project	Insert percussion into project
Assemble	Assemble the active file window (generate **.obj* file)
Build	Compile modified files only
Rebuild All	Compile all files regardless of their recent change status

Debug Menu



Go — Run program starting from the current program counter until breakpoints are hit

Free Run — Run program starting from the current program counter until **Break Execution** command (stop debugging) occurs

Run to Cursor — Run program starting from the current program counter up to the location where the cursor is anchored (applies to ICE debug mode only)

Reset — Perform hardware reset (register contents are displayed with initial values)

Step Into — Execute instructions step-by-step (with register contents updated at the same time)

Step Over — Execute instructions as in **Step Into** except for **Call** instruction, which is executed as in **Go** above.

Break Execution — Stop running program without ICE reset (program will resume running from the location where it was last stopped)

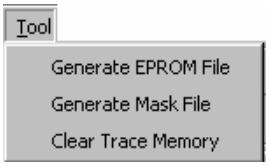
Exit Debugging — Set debugger in normal (editable) mode while ICE reset is carried out (Debugger in debugging mode sets the **Editor** window to read only)

Insert/Remove Breakpoint — Set or reset a breakpoint

Breakpoint… — Set breakpoint (under **ROM** dialog)

Clear All Breakpoints — Clear all active breakpoints

Tool Menu



Generate EPROM File — Create an EPROM program file (**.epm*) from the project

Generate Mask File — Create a mask program file (**.msk*) from the project

Clear Trace Memory — Clears all trace memory from ICE

Option Menu

Connect Define printer port connection with ICE (default is 378H)

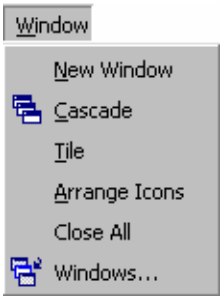
ICE Code Setting Define ICE code setting for the selected microcontroller

Project Setting Body type dynamic setting

Easy Project Setting Define an EASY project custom setting

Font… Define font for **Editor** windows (fonts for other windows are fixed & can't be changed)

Window Menu



New Window Open another window of the same file you are currently editing (for split window viewing)

Cascade Arrange all open files in cascade formation

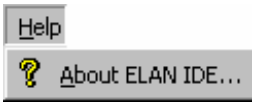
Tile Arrange all open files editing in tile formation

Arrange Icons Arrange the minimized multiple file icons (or minimize buttons) at the bottom of the **Editor** window.

Close All Close all open files

Windows… Active file windows display manipulation

Help Menu



About ELAN IDE… See information on version of the current EM60IDE Program

1.2.2.3 Toolbar

View/hide either **Standard** or **Build** toolbar, or both from **View** menu. The following are the designated function for each of the icons**:**

Standard Toolbar

1. Open file (Ctrl+O)
2. Save file (Ctrl+5)
3. Cut (Shift+Del)
4. Copy (Ctrl+C)
5. Paste (Ctrl+V)
6. Undo (Alt+Backspace)
7. Redo
8. Workspace
9. Output
10. Find (Ctrl+F)
11. Print (Ctrl+P)
12. About

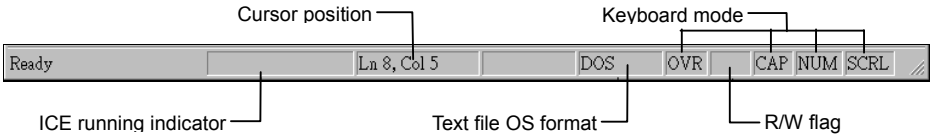
Build Toolbar

13. Assemble (Ctrl+F7)
14. Build (F7)
15. Rebuild
16. Free Run (Ctrl+F5)
17. Go (F5)
18. Run to Cursor
19. Step Over (F10)
20. Step Into (F11)
21. Break Execution (F6)
22. Exit Debugging (F8)
23. Reset (F12)
24. Insert/Remove Breakpoint (F9)
25. Clear All Breakpoints

1.2.2.4 Document Bar

The **Document** bar displays the buttons representing each of the active files in the **Editor** window. Click the button of the pertinent file that you wish to place in front of the **Editor** window to perform editing (function is similar with taskbar buttons under Windows).

1.2.2.5 Status Bar



While your project is being debugged with EM60IDE ICE, an ICE running indicator will show in the status bar.

The Cursor position indicates the cursor location within the text **Editor** window.

Text file OS format is DOS, MAC, or UNIX.

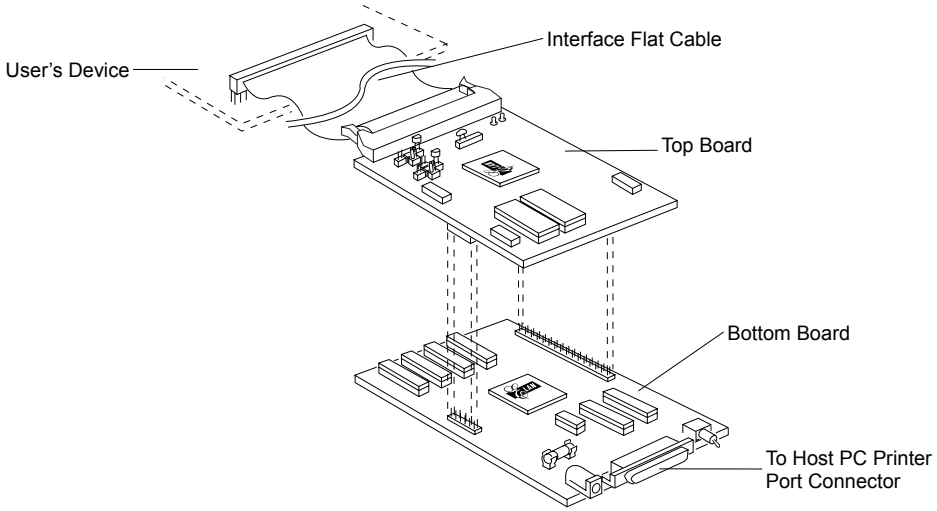
R/W flag indicates the active file Read/Write status. If Read only, “Read” will display, otherwise the field is empty.

Keyboard mode displays the status of following keyboard keys**:**

- **Insert** key – OVR is dimmed when overtype mode is off, highlighted when on
- **Caps Lock** key – CAP is dimmed when uppercase character mode is off, highlighted when on
- **Num Lock** key – NUM is dimmed when the numeric keypad calculator mode is off, highlighted when on
- **Scroll Lock** key – SCRL is dimmed when cursor control mode is off, highlighted when on

1.3 Introduction to EM60000 ICE

The EM60000 ICE is the emulation assembly that houses the EM60000 ROMless kernel chip, RAM, connectors, etc. When connected to a host PC through the printer port, it lets you download the voice code and program you are developing for an EM60000 series chip. In conjunction with the EM60000 IDE program, it then allows you to monitor, test, evaluate, debug and modify on-line, your downloaded program.

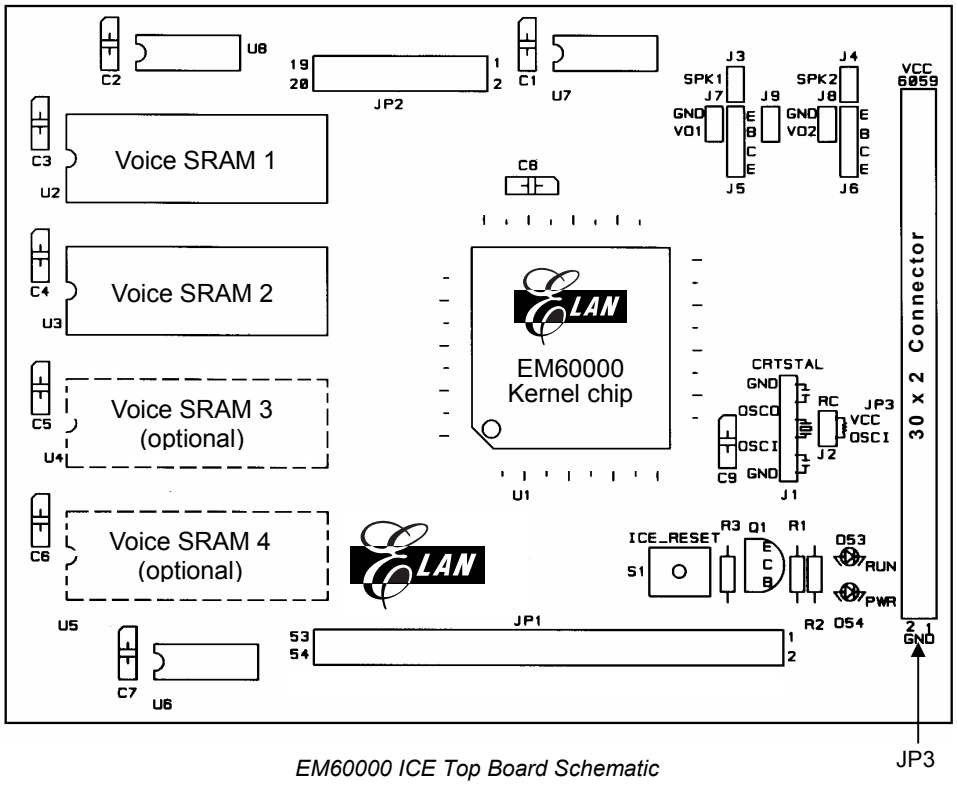


EM60000 ICE Hardware Architecture

The complete set of EM60000 ICE comprises of the following hardware modules and cables**:**

- A Main Emulation Module consisting of a top and a bottom boards. The Top Board is installed piggy-backed on the Bottom Board.
  - The top board (see figure next page) houses the EM60000 ROMless kernel chip, voice SRAMs, connectors for voice output speaker, crystal/RC, reset button, BJTs, the 60-pin connector (with I/O ports,.VCC, GND, voice output & reset pins) that connects to user's verification equipment, etc.
  - The bottom board (see figure after next page) houses the ICE interface controller, program code SRAMs, power switch, safety fuse, connectors for power and host PC, etc.
- An EM60000 ICE to host PC Interface Cable with 25-pin (printer port) D-type connectors at both ends.
- A 5VDC power adapter

1.3.1 EM60000 ICE Top Board



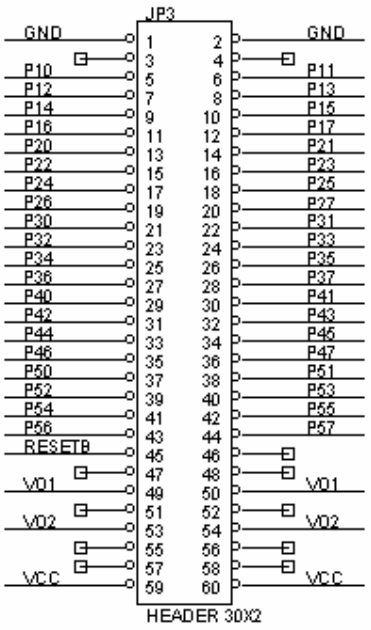
EM60000 ICE Top Board Schematic

The Top Board major components are as follows**:**

- **U1:** EM60000 ROMless kernel chip
- **U2~U5:** SRAM for voice code, only SRAM 1 & 2 (1M bytes) are currently installed as standard provision. SRAM 3 & 4 are provided only when target chip is EM60600S or when specifically requested by user
- **JP3:** 30×2 connector for VCC, GND, I/O ports, reset pin and voice output (see pin assignment next Section)
- **J1:** Crystal connection pin
- **J2:** Resistor connection pin for RC oscillator
- **J3 (SPK1) and J4 (SPK2):** Speaker connectors for VO1 & VO2 after driving BJTs
- **J5 and J6:** BJT socket for VO1 & VO2
- **J7 and J8:** By-pass resistor for BJT of VO1 & VO2
- **J9:** Jumper pins to wire-or two current D/A output pins (VO1 & VO2)
- **S1:** Push button reset switch to reset the ICE hardware (not the ROMless kernel chip)
- **D53 (RUN):** When lit, it indicates that the ROMless kernel chip is at running status
- **D54 (PWR):** When lit, it indicates that the ICE power is ON

1.3.1.1 Top Board JP3 Connector Pin Assignment

The 30×2 JP3 connector provides VCC, GND, I/O ports, reset pin and voice output pins as indicated below.



Pin 1, 2 : GND (ground signal).

Pin 5~12 : Port 1.0 ~ Port 1.7.

Pin 13~20 : Port 2.0 ~ Port 2.7.

Pin 21~28 : Port 3.0 ~ Port 3.7.

Pin 29~36 : Port 4.0 ~ Port 4.7.

Pin 37~44 : Port 5.0 ~ Port 5.7.

Pin 45 : Kernel chip Reset pin.

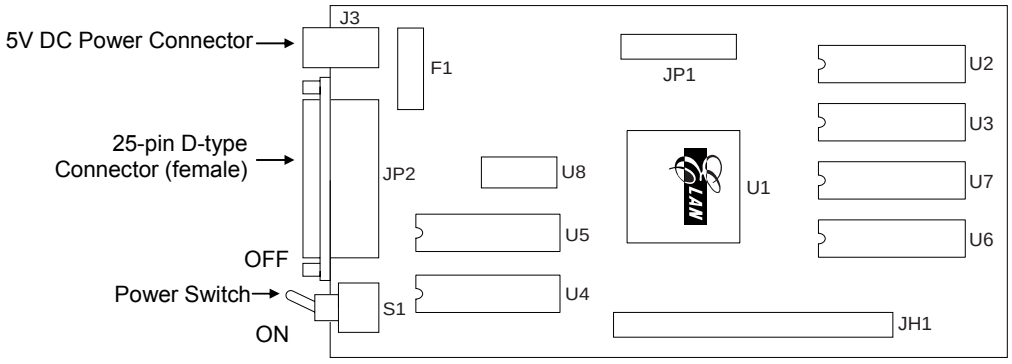
Pin 49, 50 : VO1 (D/A 1 Output).

Pin 53, 54 : VO1 (D/A 2 Output).

Pin 59, 60 : Vcc (Power pins)

EM60000 ICE Top Board JP3 Connector Pin Assignment

1.3.2 EM60000 ICE Bottom Board



EM60000 ICE Bottom Board Schematic

The Bottom Board major components are as follows**:**

- **U1:** ICE interface controller chip
- **U2, U3, U6. & U7:** Program code SRAMs
- **U4 & U5:** Trace buffer SRAMs
- **J3:** 5V power connector (positive center)
- **F1:** Fuse (1A maximum current)
- **JP2:** 25-pin D-type connector (connects ICE to PC)
- **S1:** ICE power switch

Chapter 2
System Installation

2.1 System Requirements

2.1.1 Host Computer

The EM60000 IDE requires a host that meets the following configuration**:**

- IBM PC (486-minimum, Pentium 100 or greater recommended) or compatible computers
- Win95, 98, 2000, WinME, NT, or WinXP
- 800 x 600 pixels monitor is recommended (with small fonts display is preferable)
- At least 10M byte or more free hard disk space
- At least 16M byte of RAM. 32 or more is recommended
- Mouse is optional but highly recommended

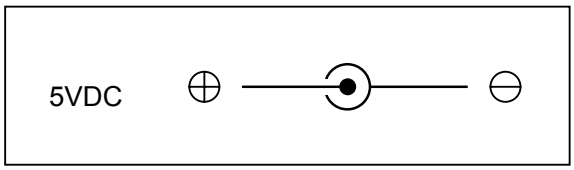
2.1.2 External Power Source

The EM60000 ICE hardware requires a single 5.0VDC power-source as provided by the power adapter that comes with the EM60000 ICE package. The power adapter has a full range input voltage range of 100 to 240 volts.

NOTE

Use of power adapter other than the one provided by ELAN Microelectronics and included with the ICE package will void warranty of the product.

A fuse (1A maximum) is installed near the DC connector for protecting the ICE from excessive current. The DC connector configuration is illustrated herewith.



EM60000 ICE DC Connector Configuration

WARNING!

Never do the following while ICE is running (EM60000 ROMless kernel chip in running status):

- *Turn power off*
- *Unplug the power adapter.*
- *Unplug the printer port connector.*

Doing so will invalidate the program and data in ICE.

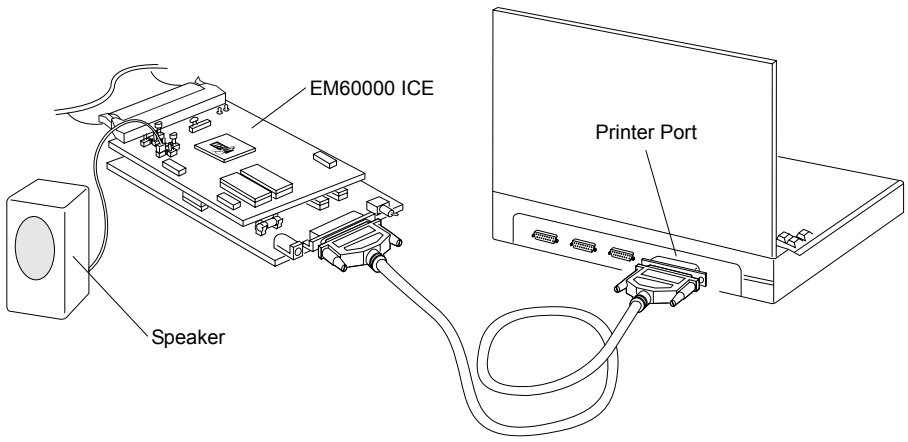
Allow the current program to complete its tasks, and then switch off power from ICE. You may proceed to disconnect any of the connectors after determining power has been switched off.

2.1.3 Audio Speaker

An 8Ω external speaker is required for audio testing of user's program on Top Board of EM60000 ICE.

2.2 System Installation

2.2.1 Connecting EM60000 ICE with Host Computer



Connecting EM60000 ICE to Host Computer

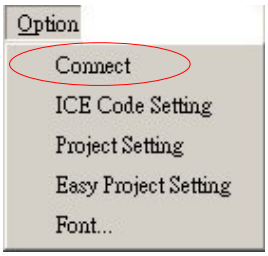
The EM60000 ICE connects to the host computer from the Top Board through 25-pin D-type connector. As illustrated above, the 25-pin male connector at one end of the cable is plugged into the JP2 connector of the Top Board. At computer side, the other end of the cable is connected to the printer port.

WARNING!

Before applying power to host or emulator board, ensure that all Interface Cable connectors are properly and tightly plugged into their respective connectors to prevent damage to the equipment.

2.2.1.1 EM60000 ICE Setup

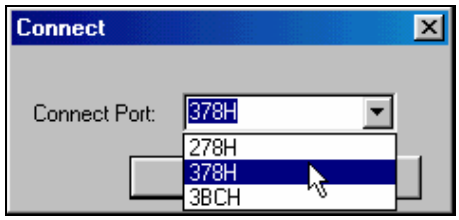
You need to configure your emulator's printer port for address compatibility with your host and define the ICE code setting that conforms to your target microcontroller. The **Option** menu is provided for this purpose.



"Option" Menu

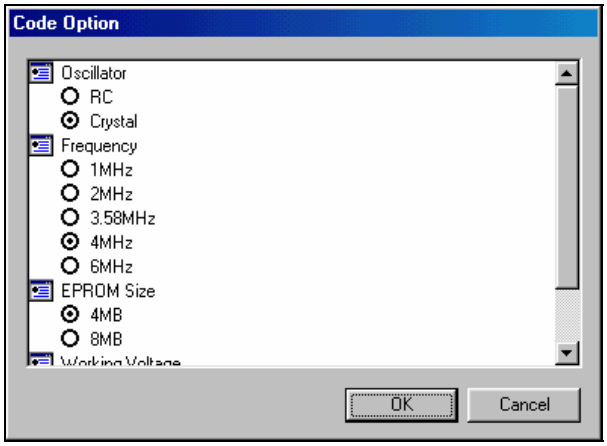
To configure printer port connection, click **Connect** command from the **Option** menu. The following dialog will display.

Select the appropriate printer port address to connect to your EM60000 ICE. Default is 378H.



"Connect" Dialog for ICE Connection to Host

To define code option, click **ICE Code Setting** command from the **Option** menu. You will be prompted to select one of the two available code options from the **Code Option** dialog. Select the appropriate code for your target microcontroller.



"Code" Option Dialog for ICE

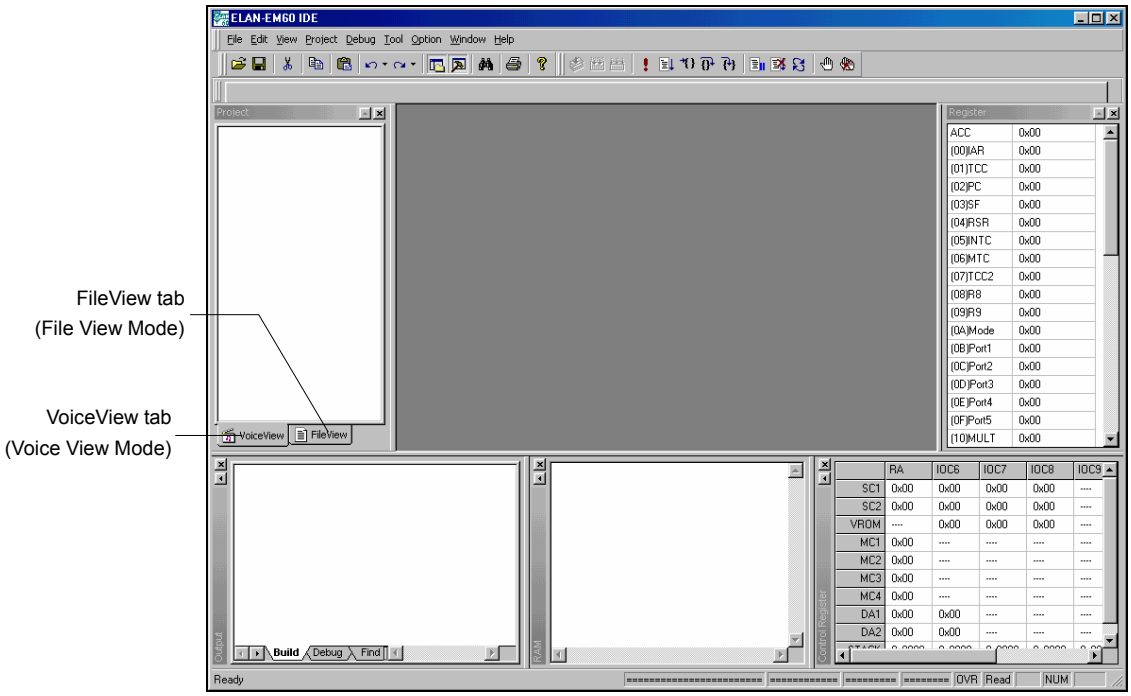
2.2.2 Installation of EM60IDE Program to Host Computer

The installation of EM60IDE Program is quite easy. Like most software installation procedure under Windows application, just follow the instruction of installer until installation is completed.

2.3 Starting the EM60IDE Program

To start EM60IDE Program, click on the EM60IDE icon from desktop or from Windows Start menu. When starting from the Start menu, click Programs, then look for EM60IDE group and click on EM60IDE icon.

Once the program is started, the main window of the program displays.



EM60IDE Program Main Window

By default, the **Project** window display in File View Mode, ready to create new project and process existing source (non-voice) files under the **FileView** tab. Go to the next chapter (Chapter 3) for the details of the EM60IDE function under File View Mode.

To process new voice data using your MIDI files, and other binary data, click the **VoiceView** tab of the **Project** window to enter into Voice View Mode of the EM60IDE Program. Voice View Mode is explained in details in Chapter 4 of this manual.

Chapter 3
Getting Started in File View Mode

3.1 Overview

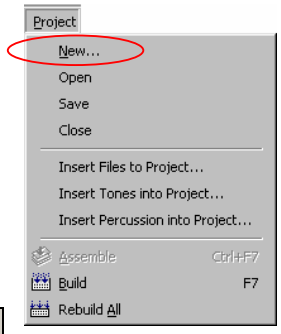
This chapter provides a step-by-step description on -

- How to create a new project under File View Mode
- How to add/remove existing source file in a project.
- How to edit, assemble, and link projects.
- How to debug the project when errors occur.
- How to generate EPROM and Mask Files from project

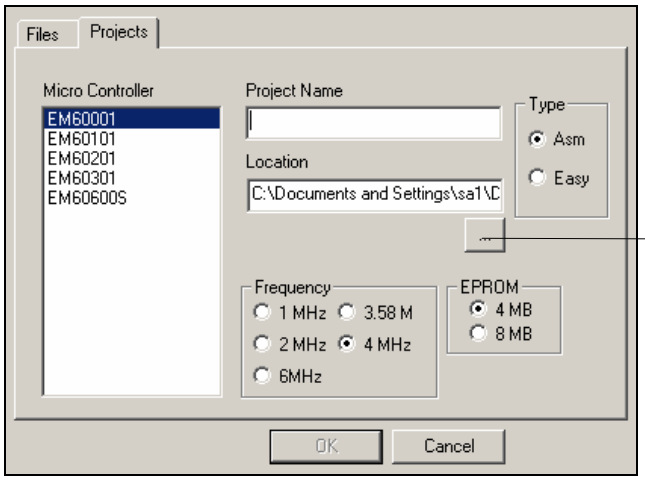
3.2 Create a New Project

To create a new project, you need to configure your project by following these steps**:**

1. From the Menu bar of the EM60IDE main window, click on **File** or **Project** menu and choose **New** command from the resulting **Pull-down** menu.
2. The **New** dialog (shown below) will then display.

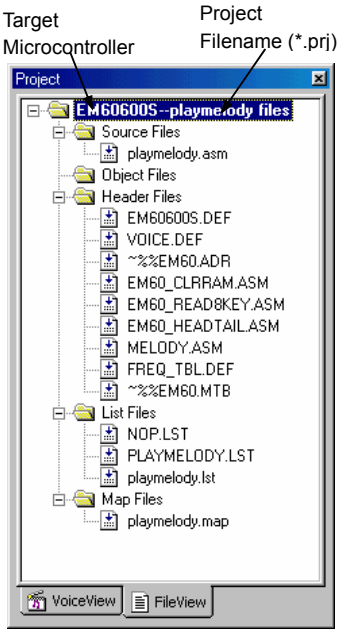


"Project" Menu



"New" Dialog Showing Project Tab for Creating New Project

3. Select **Projects** tab from dialog and assign a name for the new project in the **Project Name** box (suffix ***.prj*** will auto-append to the filename).
4. Locate the folder where you want to store the new project. You may use the **Browse** icon to find the appropriate folder.
5. Select the target microcontroller for your project from the **Micro Controller** list box.
6. Select the project type for your new project from the **Type** group box (default is ASM).
7. Assign system clock frequency by clicking on one of the options provided in the **Frequency** group box. Default is 4 MHz.
8. Assign desired size for your demo EPROM by clicking on one of the two options provided in the **EPROM** group box. When 4MB (default) is selected, the resulting EPROM data file is suffixed with ***.v40*** (additional file with ***.v41*** suffix is created if file size exceeds 512K). When 8MB EPROM is selected, the resulting data file is suffixed with *v80*. 27C040 EPROM (for 512Kb) and 27080 EPROM (for 1024Kb) are recommended.
9. Click **OK** button after confirming all your choices and inputs.

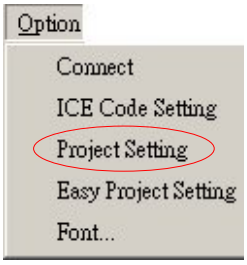


File View Mode "Project" Window

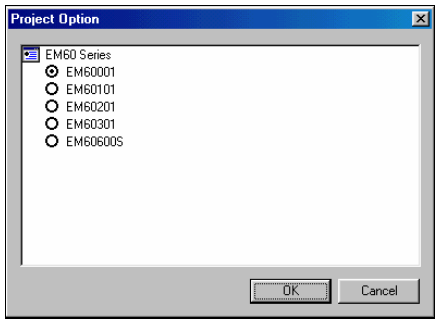
The new project is created with the defined project name and microcontroller you have selected displayed on top of the **Project** window (under **File View** tab), which now shows all the source program components. Click **File View** tab if it is not active.

3.2.1 Changing the Target Microcontroller Halfway into the Project

When you want to change the target microcontroller you have previously selected for your project, click on **Option** menu and choose **Project Setting** command. The **Project Option** dialog (shown below), listing all the available microcontrollers will then display



"Option" Menu



"Project Option" Dialog for Changing Target Chip

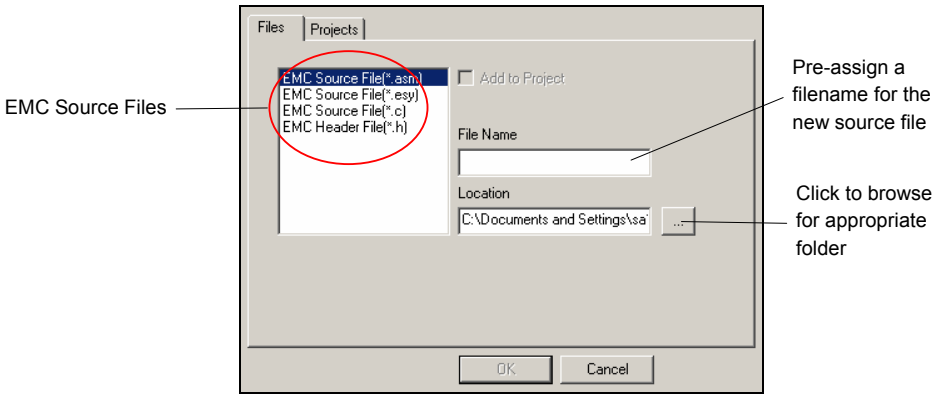
3.3 Add and Remove Source Files from/to Project

You can either insert existing source files into the new or existing project, or create new ones from within EM60IDE text Editor and automatically insert them into the new project.

3.3.1 Create New Source File for the Project

If your source file is yet to be created, you can take advantage of the **New** dialog (by clicking **New** command from the File menu) to create your new source file filename and use the EM60IDE text editor to compose its content.

1. Click the **Files** tab of the **NEW** dialog and select the type of source file you want to create from the **EMC Source File** list box, i.e., **.asm* (default) for assembly file; **.esy* for EASY file, **.c* for C Language file, and **.h* for header file.



"New" Dialog Showing File Tab for Creating a New Source File

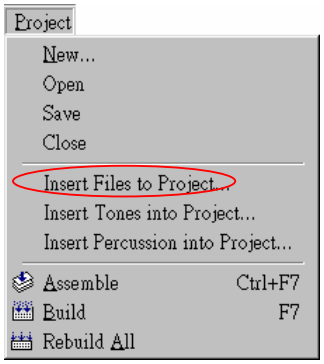
2. Assign a filename for the new source file in the **File Name** box.
3. Locate the folder where you want to store the new source file in your disk. You may use the **Browse** icon to find the appropriate folder.

4. Check **Add to Project** check box (default) if you want to automatically add the new file into your project. Otherwise clear the check box.
5. Click OK button after confirming your inputs. You will be prompted to start writing the newly defined source file in the **Editor** window.

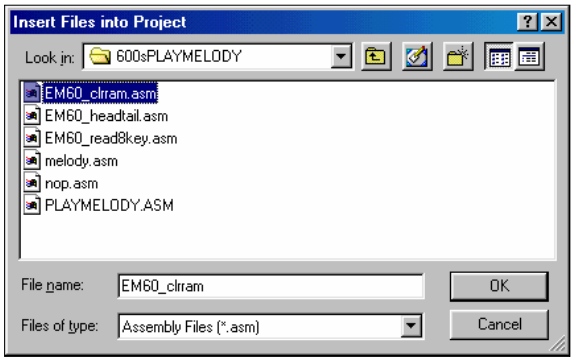
3.3.2 Add Existing Source Files to the New Project

If your source file is ready, you can immediately insert it into your new project.

1. From the Menu bar, click on **Project** menu. Choose **Insert Files to Project** command from the resulting pull-down menu and the **Insert Files into Project** dialog will display.



"Project" Menu



"Insert File into Project" Dialog

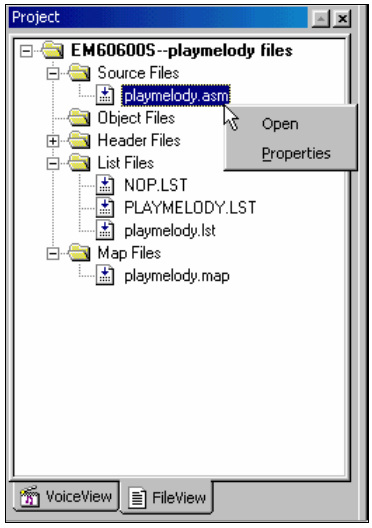
2. Browse and select the file or multiple select the files you wish to insert into the new project.
3. Click **OK** button after confirming your choice.

3.3.3 Open Source File or Display its Path

You can open source file into **Editor** window for editing from **Project** (FileView) window. To do this, double click on the source file and the file is automatically opened. You can also open source file by right clicking on the file you want to open. Then click the **Open** command from the resulting shortcut menu.

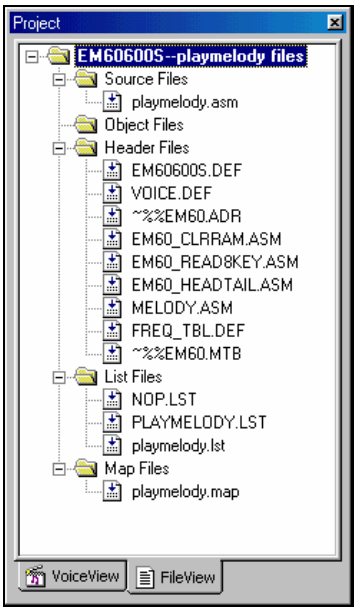
Clicking the **Properties** command of the shortcut menu will provide the path to the source file location, e.g.-

Filename: C:\BCG464\BCGDevStudioExample464\AsmSrc\playmelody.asm



Opening Source File from "Project" Window

3.3.4 Deleting Source Files from Project

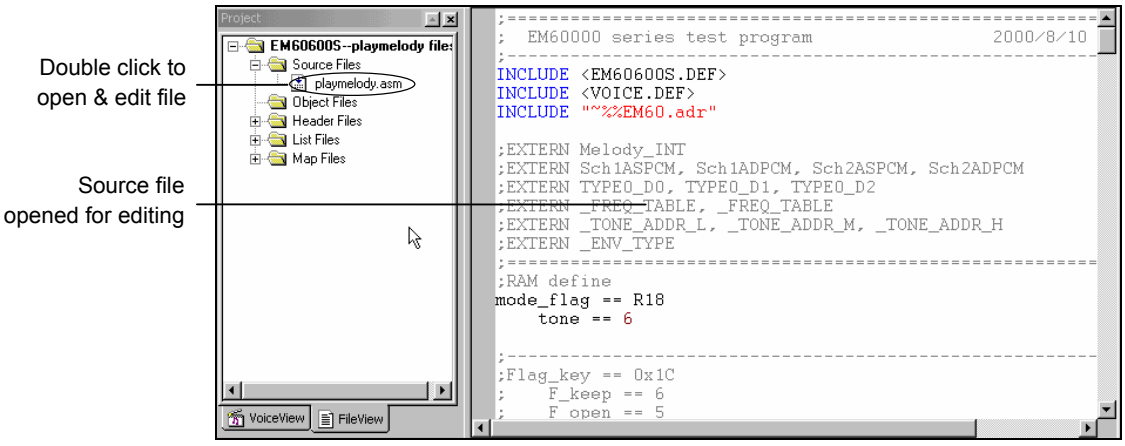


From the **Project** window, select the file(s) you wish to delete. Then press the **Delete** key from your keyboard.

"Project" Window

3.4 Edit a Source File From and Within Project

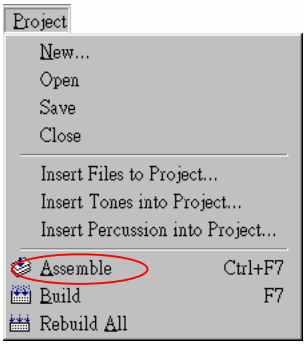
You can edit source files that are already inserted in the project. To do so, double click the source file you wish to edit from the **Project** window and the file will open in the **Editor** window of the EM60IDE Program



Editing Source File From and Within Project

3.5 Assemble and Link the Project

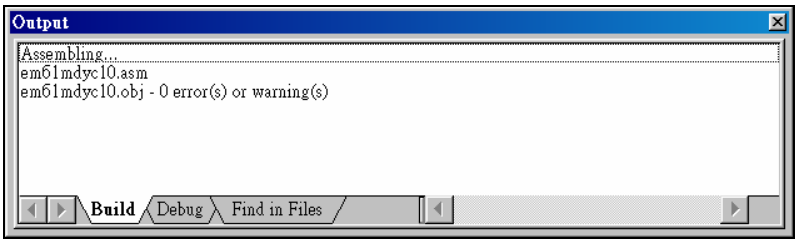
With your source file(s) inserted into the project, you are now ready to compile your project.



"Project" Menu

3.5.1 Assemble Files to Generate Object File

Be sure you have your source (**.asm*) file(s) inserted in the Project window. Then, click **Assemble** (Ctrl+F7) from the **Project** menu. The EM60IDE will then assemble the source files and generate an object file (**.obj*, with your project name as filename). The object file is automatically saved in the same folder where your source files are located. Status of the assembly operation can be monitored from the **Output** window as shown below.



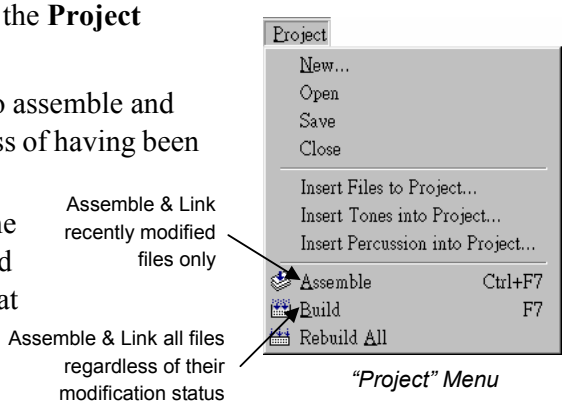
Output Window Showing Successful Assembly Compilation

If error is detected during compilation, pertinent error message will also display in the **Output** window. Double click on the error message to link to the source of error from corresponding program text line in the **Editor** window.

3.5.2 Assemble and Link Files to Generate Binary File

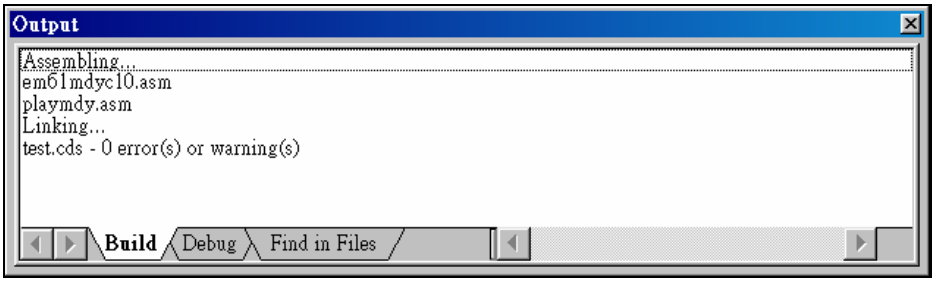
With the source file(s) shown in the **Project** window –

- Click **Rebuild All** command to assemble and link all inserted files regardless of having been recently modified or not.
- Click **Build** command from the **Project** menu to assemble and link only those source files that were recently modified.



"Project" Menu

EM60IDE will assemble and link all source files to generate binary (*.*bin*) files using your project name as filename. The binary file is automatically saved in the same folder where your source files are located. Status of the assembly operation can be monitored from the **Output** window as shown below.



Output Window Showing Successful Assembly & Link Compilation

If error is detected during compilation, pertinent error message will also display in the **Output** window. Double click on the error message to link to the source of error from corresponding program text line in the **Editor** window.

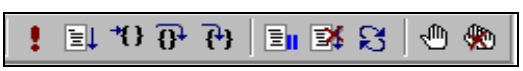
Output Window Showing Assembly & Link Compilation Errors

Modify source files to correct the errors and repeat assemble and link operations.

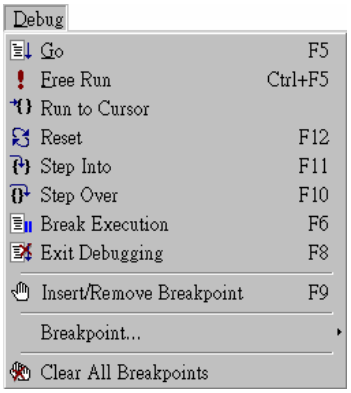
3.6 Debugging a Project

With the source files deprived of its errors, you are now ready to debug the files. Be sure the EM60000 ICE is properly connected (see Chapter 2; *System Installation*) to your computer.

Debugging commands are available from the **Debug** Menu as well as from the EM60IDE Program Toolbar.



Toolbar for Debugging Commands



"Debug" Menu

3.6.1 Defining Breakpoints and Program Counter

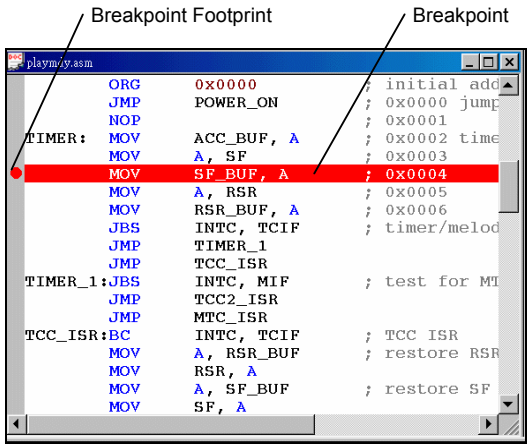
Breakpoints are a combination of addresses and loop counter. A program stops when one of the defined addresses matches with the contents of the program counter and its relative loop counter decreases to zero. When the program stops, the contents of program counter, all the registers and RAM are read from the emulation chip and displayed in their respective windows. The register values that have been recently changed (prior to breakpoint execution) are expressed in red. You can set as many breakpoints as you want.

NOTE

Breakpoint may be set at any address of the 8K Program ROM. When the loop size is larger than "1," it should be put on loop counter sets. However, there should be only 4 loop counter sets in a debugging session. Loop counter value is from 0 to 255.

3.6.1.1 Breakpoint Setup

To assign a breakpoint, position cursor at any point of the line where a breakpoint is going to be set, and then press **F9** key. Observe the line highlighted in red with a red footprint (dot) to its left margin.


Breakpoint Footprint
Breakpoint
playmidy.asm
ORG 0x0000 ; initial add
JMP POWER_ON ; 0x0000 jump
NOP ; 0x0001
TIMER: MOV ACC_BUF, A ; 0x0002 time
MOV A, SF ; 0x0003
MOV SF_BUF, A ; 0x0004
MOV A, RSR ; 0x0005
MOV RSR_BUF, A ; 0x0006
JBS INTC, TCIF ; timer/melod
JMP TIMER_1
JMP TCC_ISR
TIMER_1:JBS INTC, MIF ; test for MT
JMP TCC2_ISR
JMP MTC_ISR
TCC_ISR:BC INTC, TCIF ; TCC ISR
MOV A, RSR_BUF ; restore RSR
MOV RSR, A
MOV A, SF_BUF ; restore SF
MOV SF, A

Active Source File with a Defined Breakpoint

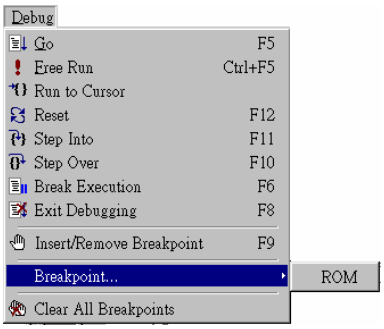
You can also click on the **Insert/ Remove Breakpoint** icon (hand shape) on the toolbar, or choose **Insert/Remove Breakpoint** command from the **Debug** menu to set the breakpoint.

Likewise, the defined breakpoint is cleared if **F9** or the hand icon is clicked the second time with the cursor positioned on the defined breakpoint. To clear all existing breakpoint, click **Clear All Breakpoints** command from **Debug** menu or its icon from **Toolbar**. Note that this method of clearing a breakpoint will also remove the breakpoint footprint. To remove breakpoint without deleting its footprint, use the **ROM** dialog as explained below.

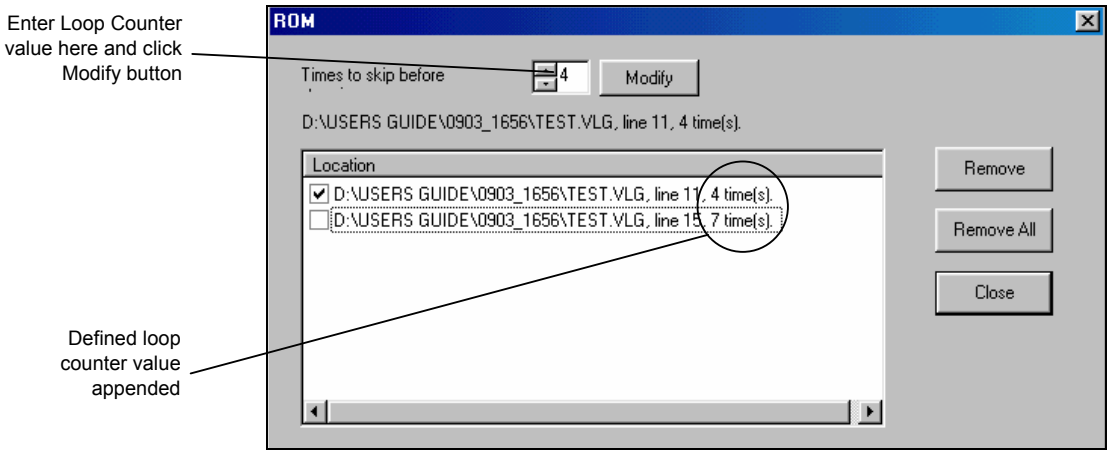
3.6.1.2 Breakpoint Loop Counter Setup

To set the number of times to skip a breakpoint before stopping**:**

1. Click on **Breakpoint** command from the **Debug** menu. Then click on the **ROM** sub-menu, the **ROM** dialog will display (see figure below).



Accessing ROM Dialog from "Debug" Menu

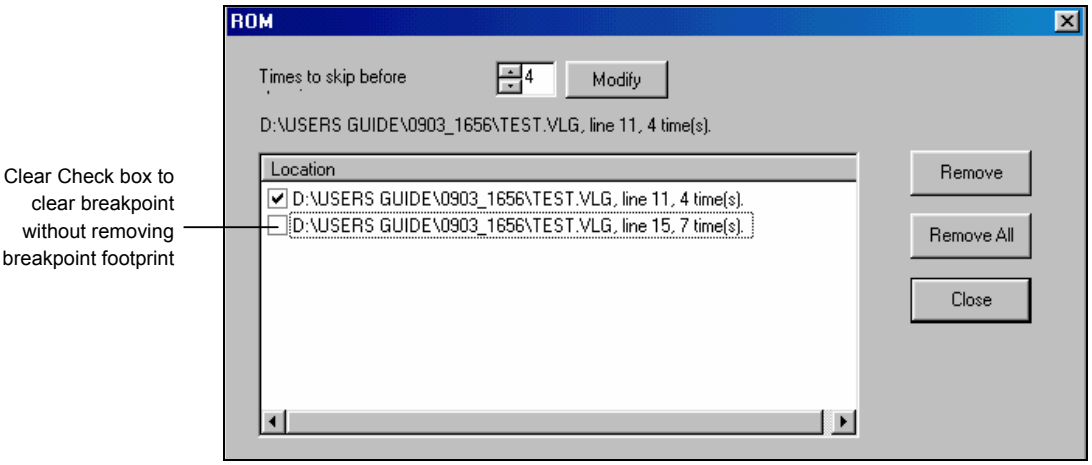


"ROM" Dialog Showing Two Defined Breakpoints

2. From the **Location** list box, select (enable checkbox) the source file(s) you wish to assign a loop counter.
3. Go to the **Times to skip before** edit box and directly enter the desired loop counter value (0 ~ 255; default is 0) or use the spin button to spin for the desired value.
4. Click on the **Modify** button to apply the value. Observe the loop counter value being appended at the end of the corresponding breakpoint.

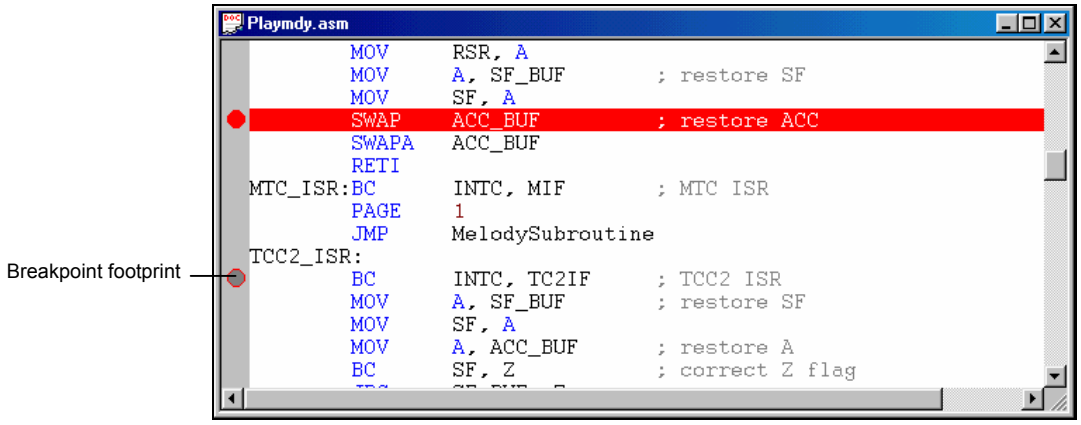
3.6.1.3 Enable/Disable Breakpoint from ROM Dialog

To disable a breakpoint without removing its footprint (a useful feature when you want to clear breakpoint temporarily), clear the check box of the selected breakpoint.

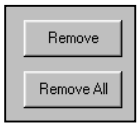


"ROM" Dialog Showing a Breakpoints Cleared without Removing its Footprint

Observe the corresponding breakpoint being cleared (highlight disappears) but its footprint remains.



Active Source File Showing a Cleared Breakpoint with its Footprint Intact



To remove a particular breakpoint and its footprint, select the breakpoint from the Location list box and click on the Remove button. To totally remove all breakpoints and their respective footprints, click Remove All button.

You can also clear all breakpoints and footprints by clicking **Clear All Breakpoints** from the **Debug** menu or pertinent icon from **Toolbar**.

3.6.2 Debugging Tools

Start and perform debugging with the following commands from the **Debug** menu or their pertinent icons from **Toolbar:**

F5

Go - Run program starting from the current program counter until breakpoints are hit and breakpoint address executed.

Ctrl+F5

Free Run - Run program starting from the current program counter until **Break Execution** command occurs. All defined breakpoints are ignored during free run.

F11

Step Into - Execute instructions step-by-step (with register contents updated at the same time).

F10

Step Over - Execute instructions as in **Step Into** command except for Call instruction, which is executed as in **Go** command (subroutine skipped).

Run to Cursor - Run program starting from the cur rent program counter up to the location where the cursor is anchored (applies to ICE debug mode only).

F6

Break Execution - Stop running program without ICE reset (program will resume running from the location where it was last stopped).

F12

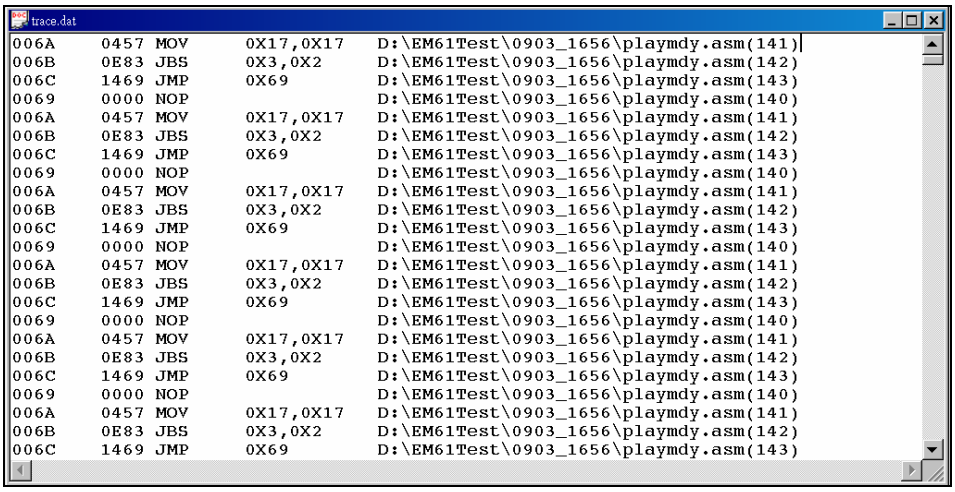
Reset - Perform hardware reset (register contents are displayed with initial values). ICE will return to its initial condition.

F8

Exit Debugging - Set Debug in normal (editable) mode and ICE reset is carried out (Debug in debugging mode sets the **Editor** window to read only)

3.6.2.1 View Trace Buffer

To view the debugging history in the trace buffer, click the **Trace buffer** command from the **View** menu. All the available debugging history will be displayed in the **Editor** window as shown below. Note that only the executions of the **Go**, **Free Run**, and **Run to Cursor** commands are recorded.

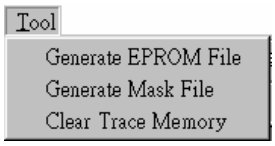


Trace Buffer Display

To close trace buffer, click the **Trace buffer** command again.

3.7 Generate EPROM and Mask Files from Project

EM60IDE Program allows you to generate EPROM and Mask files for demo board verification application.



"Tool" Menu

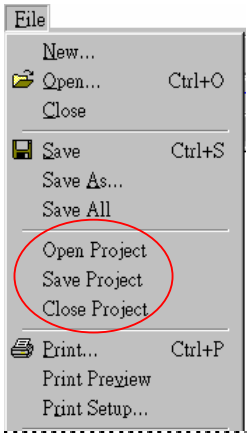
To generate EPROM (**.epm*) and Mask **.msk*) program files from your active project, go to the **Tool** menu .

- Click **Generate EPROM File** command to create EPROM program file. Filename will be the same as project name but with ***.epm*** suffix. The EPROM file should not exceed 16K bytes. 27C256 or 27C512 EPROM are recommended.
- Click **Generate Mask File** command to create **.msk* program file. Filename will be the same as project name but with ***.msk*** suffix

The location of the generated EPROM/Mask file will be displayed in the **Output** window. By default, the files are placed in the same folder where your project file is located.

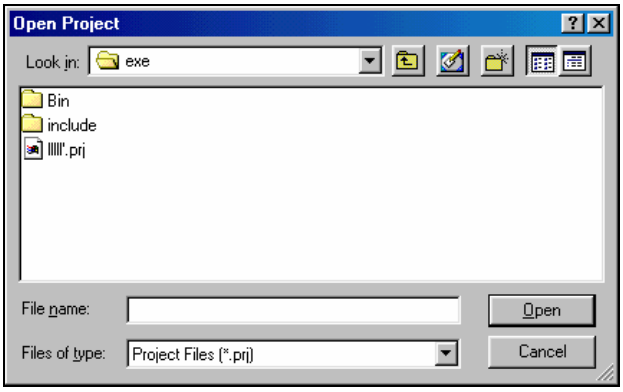
3.8 Save, Open, and Close Projects

To save your new project, click **Save Project** from the **File** menu. The project is automatically saved in the same folder where your EM60IDE Program is located.



"File" Menu

To open an existing project; click **Open Project** from the **File** menu. You will be prompted to select the project you want to open from the **Open Project** dialog.



"Open Project" Dialog

To close the active project, click **Close Project** command from the **File** menu.

Chapter 4
Getting Started in Voice View Mode

4.1 Overview

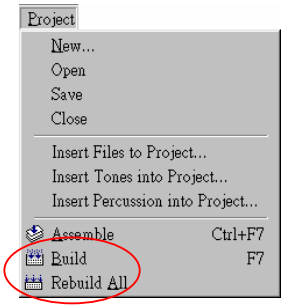
This chapter provides a step-by-step description on -

- How to add/remove source file in a project.
- How to edit, assemble, and link the projects.
- How to debug the project when errors occur.
- How to generate EPROM and Mask Files from project.

4.2 Creating an Entirely New Voice Project

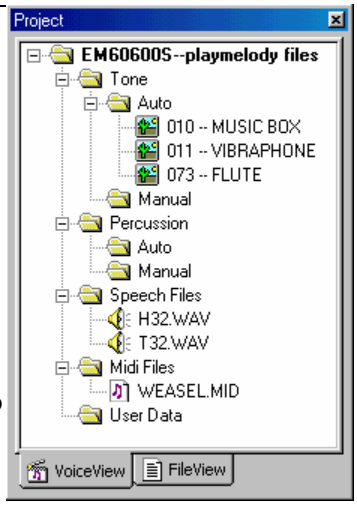
To create an entirely new voice project, repeat the procedure specified in Section 3.2 of Chapter 3. The newly created voice project will be displayed in the **Project** window (under **Voice View** tab) with its tree structures, i.e., **Tone**, **Percussion**, **Midi Files**, **Speech Files**, and **User Data** nodes (see figure at right below). The defined project name and microcontroller appear on top of the window. You are now ready to insert your source files and start compiling the voice data files

4.3 Compiling Project



"Project" Menu

Before you could start compiling you need to initially insert (see next section) the required midi and binary (or table) source files into **Midi Files**, **Speech Files**, and **User Data** nodes respectively. With these nodes filled up, you may execute the automatic compiling commands (**Build** or **Rebuild All**) from **Project** menu and the EM60IDE starts to automatically collect the tone and percussions instruments from MIDI file and display them under "Auto" of **Tone** and **Percussion** nodes respectively.



"Project" Window after Auto Compilation

NOTE

*Use **Build** command to compile the recently modified instrument files only. Use **Rebuild All** command to compile all instrument files disregarding its recent change status.*

You can also manually select your own instrument files from your tone and/or percussion library and insert them into **Tone** and/or **Percussion** nodes in addition to or independent from midi supplied files. See Section 4.3.4 and 4.3.5 on how this is done. The same **Build** or **Rebuild All** command is used to compile the manually inserted instruments.

4.3.1 Building Voice Data

Whenever the **Build** or **Rebuild All** command is executed, the EM60IDE will automatically accomplished the following tasks. Note that all the generated files will bear the same filename as that of the defined voice project name and are placed in the same folder where your project is located**:**

1. Collect the instruments from the MIDI files or the manually inserted instruments and translate them into melody codes.
2. Translate the text data (**.bin*) and collects the binary (**.tbl*) files
3. Create Voice Data File for downloading into the EM60000 ICE. The automatically created file is suffixed with **.vd*.
4. Generate MIDI/data start address information, which can be included in your assembly source programs.
5. Create EPROM Data File for the type of EPROM you have defined in the **New** dialog setup for creating new project (see Section 4.2).

   If you have chosen the 4MB EPROM, the generated EPROM data file will be suffixed with **.v40* when the file size is within 512K. When file size exceeds 512K, a second data file will be created with a **.41* suffix.

   If the 8MB EPROM is chosen, the data file will be suffixed with **.v80*.

4.3.1.1 Melody Code Header

While compiling, the EM60IDE collects the melody and data codes and writes them to a voice data file. The melody codes has header containing information about the sampling and tempo. The first two bytes are the tempo information.

The 3 MSB of the first byte is the MTC pre-scale value. You can write this value to MSR2 ~ 0 of prescale control register.

The second byte is the MTC timer value. You can just write this value to the MTC timer counter register. By doing so, the correct MTC interval (corresponding to tempo speed) will be set up.

4.3.1.2 How EM60IDE Translate Your MIDI File

- The EM60IDE translator will skip the "channel" setting in the MIDI file. Hence, the "channel" setting in MIDI file are ignored and has no influence whatsoever on the hardware channels of EM60000 series IC.
- Because the EM60000 series IC are only equipped with eight hardware channels, the translator can easily collect notes and assign them to the IC hardware channels automatically. The only thing you need to do is to check and be sure that, at most, only eight notes are active at the same time. As the translator support polyphony (chord), these notes may be located in the same or different tracks (staffs). If more than eight notes are active at the same time, the translator will fail and an error message will display.
- If there are instruments in the MIDI file that are not supported by the EM60IDE, the translator will show a warning message and replace all the unsupported instruments with the default instrument (Insert Tone List # "000"; Acoustic Grand Piano).

  To view all the instruments available for your application, do the following**:**
  - Click **Insert Tones into Project** command from the **Project** menu. The **Insert Tone** dialog will display with the list of all available tone instruments (see Section 4.3.4 for more detail).
  - Click **Insert Percussion into Project** command from the **Project** menu. The **Insert Percussion** dialog will display with the list of all available tone instruments (see Section 4.3.6 for more detail).
- When editing the common MIDI file on PC, percussion can be identified by setting the "channel" parameter to "10," and use different notes to represent different percussion instruments. This is also true in the EM60IDE translation. The rule is the same but with some restrictions as described below**:**
  - The percussion must have its own exclusive track. In the MIDI edit tool, use a track to place the percussion and set the "channel" parameter to "10". Do not insert any other instrument.
  - The translator will look for the tracks that have its "channel" parameter set to "10." All the notes in these tracks will be recognized as percussion instruments.
  - The MIDI events, such as Program Change, Volume, etc., will be omitted.

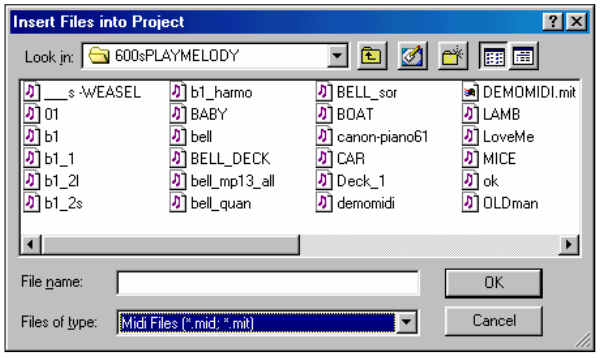
4.3.2 How to Insert MIDI files into MIDI Node

NOTE

Your MIDI file should meet the following conditions in order for the EM60IDE to convert them accurately:

1. *The MIDI file should be suffixed with ".mid" or ".mit" extension.*
2. *The EM60IDE can only recognize the General MIDI Format 1 of the MMA specifications. Formats 0 and 2 are not supported.*
3. *There is no limit to total number of tracks. EM60IDE will automatically assign the track to IC channels accordingly.*
4. *EM60IDE supports polyphony (chord). Each track may have more than a pair of "Note On" and "Note Off."*

To insert one or more MIDI files to your voice project, click the **Insert Files to Project...** command from the **Project** menu. From the resulting **Insert Files into Project** dialog, browse for "Midi File (**.mid*)" file folder and select the one you want to insert into the voice project and click **OK** button. The selected MIDI file will show under the **Midi Files** node of the **Project** window.

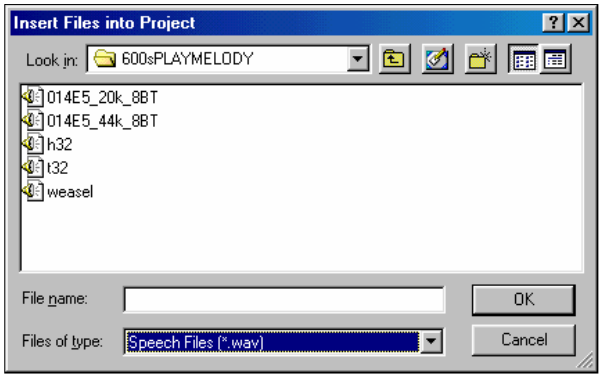


"Insert Files into Project" Dialog Showing MIDI Files

To remove MIDI file(s) from the node, select the file and press the **Delete** key.

4.3.3 How to Insert Speech Files into Speech Node

To insert one or more wave files to your voice project, click the **Insert Files to Project...** command from the **Project** menu. From the resulting **Insert Files into Project** dialog, browse for "Speech File (**.wav*)" file folder and select the one you want to insert into the voice project and click **OK** button. The selected wave file will show under the **Speech Files** node of the **Project** window.

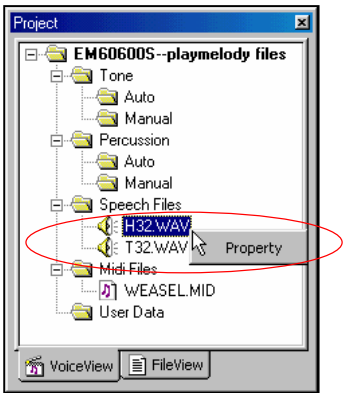


"Insert Files into Project" Dialog Showing Speech Files

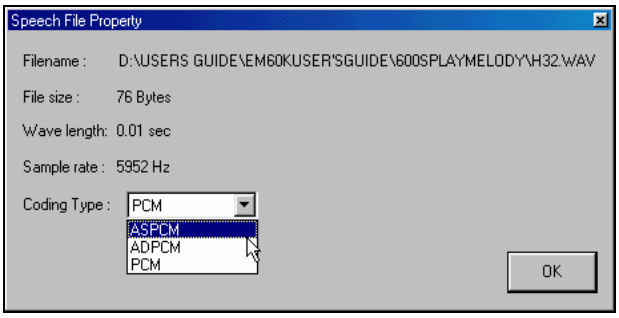
To remove wave file(s) from the node, select the file and press the **Delete** key.

You can also set the coding type (PCM, ASPCM or ADPCM) for the selected speech wave file by-

1. Right-clicking the filename ("H32.WAV" in the example at right) under the **Speech Files** node.
2. Click the **Property** command from the resulting shorcut menu and the EM60IDE will display the **Speech File Property** dialog of the selected speech wave file.



Setting Speech File (.wav) Coding Type*

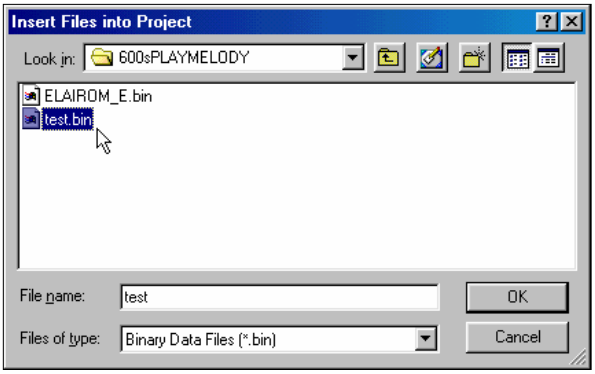


"Speech File Property" Dialog

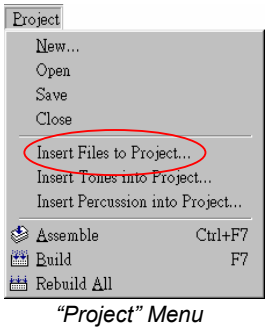
The dialog box provides information on filename path, file size in byte, waveform length in second, sampling rate, and the current coding type setting. You may change the coding type by clicking the arrow of the **Coding Type** drop-down list box.

4.3.4 How to Insert User Data into User Data Node

User Data node is where your table (*.*tbl*) and/or binary (*.*bin*) files are inserted for inclusion into your voice project to support instrument playback function of the EM60000 series sound processors.



*"Insert Files into Project" Dialog Showing *.bin Files*

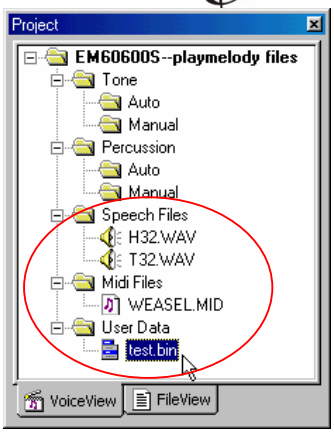


"Project" Menu

To insert User Data, click the **Insert Files to Project**... command from the **Project** menu. From the resulting **Insert Files into Project** dialog, browse for the (*.tbl*) or (*.bin*) folder and select the file you want to insert in the voice project.

Press the OK button in the dialog to apply selection. The selected data files will be shown under the **User Data** node of the Project window.

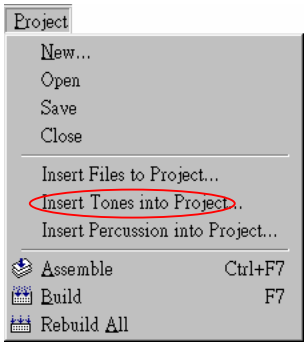
To remove data file(s) from the node, just select the file and press the **Delete** key.



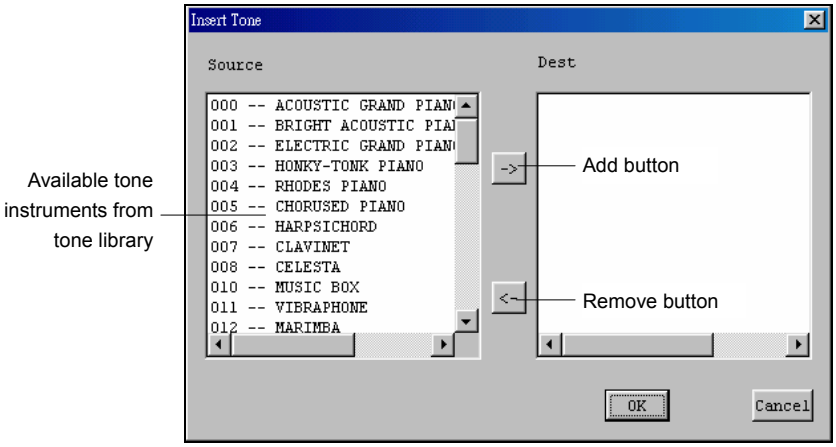
"Project" Window after Inserting Speech, MIDI, & Data Files

4.3.5 How to Insert Instrument Files into Tone Node Manually

To manually select instruments from tone library and insert them under "Manual" of **Tone** node, click the **Insert Tones into Project…** command from **Project** menu. Select the desired instruments from the **Source** box of the resulting **Insert Tone** dialog and click the **Add** button to insert the selected instruments into the **Dest**ination box. The selected instruments are then displayed under the "Manual" node after you click the **OK** button.



"Project" Menu



"Insert Tone" Dialog

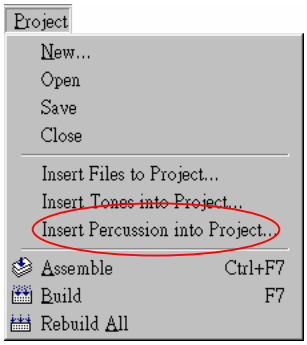
To remove tone instruments from the "Manual" node, you may return to this dialog and select the instrument you want to delete and click the **Remove** button. Or select the unwanted instruments directly from under the 'Manual" node and press **Delete** key.

WARNING!

You cannot directly remove the instrument files displayed under "Auto" node. You have to get rid of the unwanted file from your sourced midi file and execute Build or Build All command again.

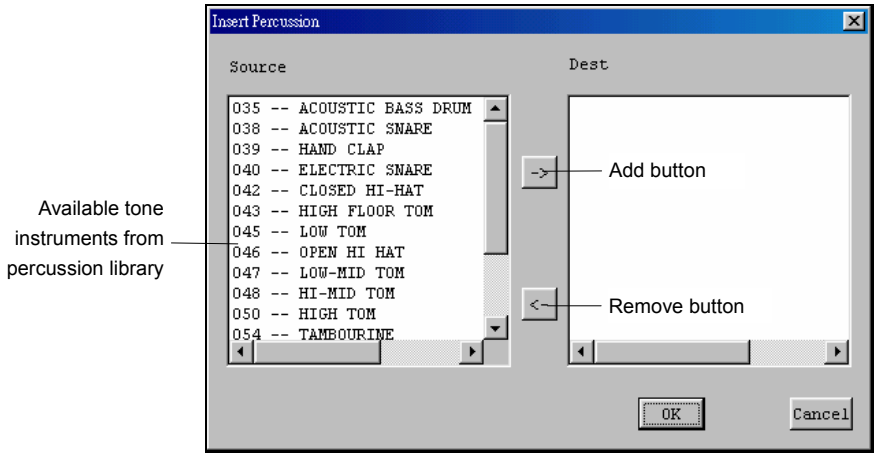
4.3.6 How to Insert Instrument Files into Percussion Node Manually

Percussions can be very helpful when used to augment instruments not available from you MIDI files and provide special effect when required. EM60IDE will record and add percussions to the melody table during compilation.

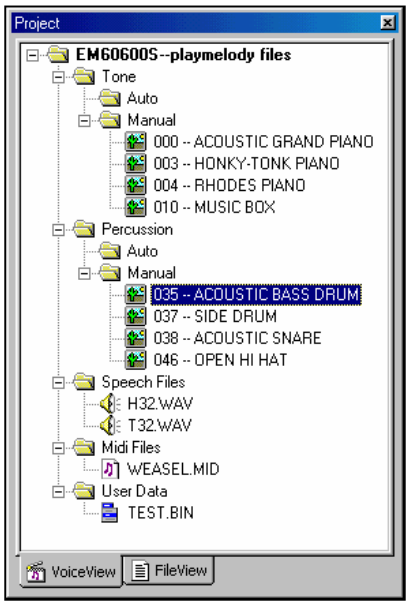


"Project" Menu

The procedure of inserting percussions under "Manual" of **Percussion** node is practically the same as it is done under **Tone** node (see Section 4.3.3). Their only difference is use the **Insert Percussion into Project…** command to display its pertinent **Insert Percussion** dialog as displayed below.



"Insert Percussion" Dialog



"Project" Window after Inserting Tone & Data Files Manually

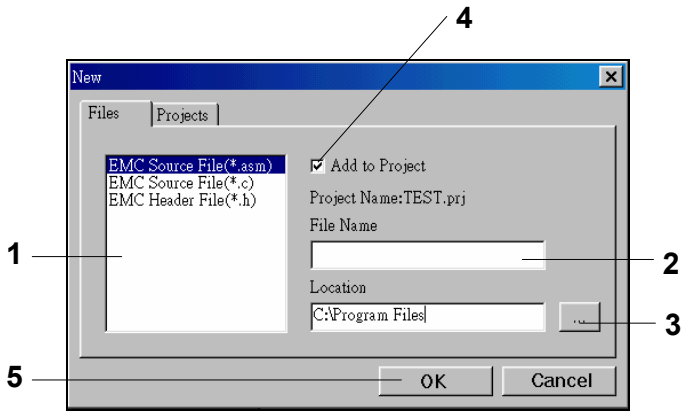
Like tone instruments, you can remove selected percussions under the "Manual" node but not from "Auto" node. To delete percussion files from "Manual" node, follow the procedure stated for Tone node above.

Chapter 5
Assembler and Linker

5.1 Introduction

If your assembly file is yet to be created, you can take advantage of the EM61IDE's **New** dialog (by clicking **New** command from the File menu) to create your new assembly file filename and use the EM61IDE text editor to compose its content.

1 Click the **Files** tab of the **NEW** dialog and select the *EMC Source File(.asm)* as your source file type from the **EMC Source File Type** list box.



"New" Dialog Showing File Tab for Creating a New Source File

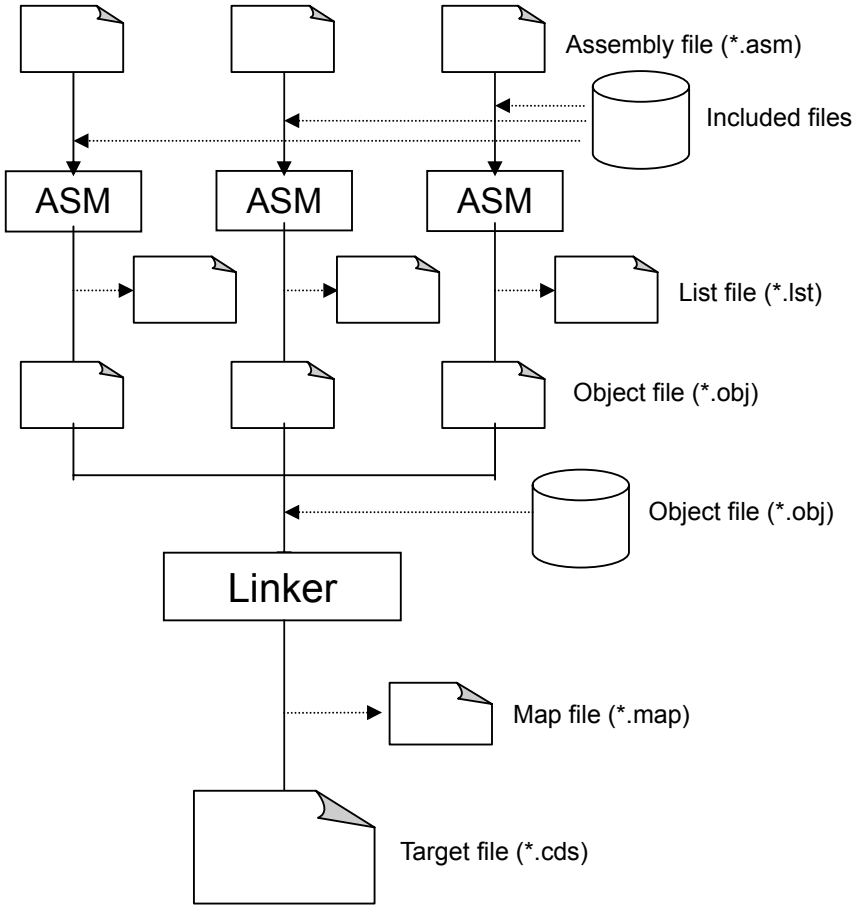
2 Assign a filename for the new assembly source file in the **File Name** box.

3 Locate the folder where you want to store the new assembly source file in your disk. You may use the **Browse** icon to find the appropriate folder.

4 Check **Add to Project** check box (default) if you want to automatically add the new assembly file into your project. Otherwise clear the check box.

5 Click OK button after confirming your inputs. You will be prompted to start writing the newly defined assembly file in the **Editor** window.

5.2 Assembler and Linker Process Flow



Assembler and Linker Process Flow Chart

5.3 Assembler Conventions

5.3.1 Statement Syntax

The statement syntax is shown in the following format:

```
[label[:]] operator [operand] [,operand] [; comment]
```

Where:

[ ]: Characters enclosed in brackets are optional

Space & Tab: Used to separate elements from each other

Label: The colon (**:**) is optional and is followed by one or more spaces or tabs.

A label may consists of alphabets, digits, and underscores (Ex**:** A~Z a~z 0~9 _).

The digit character (0~9) must not be the first character of a label. Only the first 31 characters are recognized

Operator: An assembler instruction or directive. Directives provide the direction to the assembler, specifying the manner with which the assembler is to generate object code at assembly time.

Operand: There can be one or more operands separated by commas.

Comment: A comment can be a line comment or block comment.

Line comment is preceded by a semi-colon (**;**).

Block comment begins with "/*" and ends with "*/".

NOTE

Label, Operator, Operand, and Comment are NOT case-sensitive.

5.3.2 Numeric Presentation

Type	Type 1	Type 2	Type 3
Decimal	0D<digits>	<digits>D	<digits>
Hexadecimal	0X<digits>	<digits>H	-
Octal	0Q<digits>	<digits>Q	-
Binary	0B<digits>	<digits>B	-

Note:

1. In Type 2, hexadecimal presentation, if the first character is not a digit, a zero (0) must be added as a prefix.
2. If conflict occurs between Type 1 and Type 2, Type 1 will take precedence, i.e., the assembler will use the Type 1 presentation.

5.3.3 Arithmetic Operations

All the assembler arithmetic operations are evaluated during assembly period. Errors will be generated if a value is not successfully calculated while assembling. The assembler use integer operation and will translate floating point value to integer automatically.

5.3.3.1 True and False

The assembler uses 0xFF as TRUE and 0x00 as FALSE.

5.3.3.2 Summary of the Operators (In the Order of Priority)

1. Parentheses: **( )**
2. Unary operators:
   - **!** Logical NOT
   - **~** Complement
   - **–** Unary minus
3. Multiplication arithmetic operators/shift operators:
   - **×** Multiplication
   - **/** Division
   - **%** Modulo
   - **<<** Logical shift left
   - **>>** Logical shift right
4. Addition arithmetic operators:
   - **+** Addition
   - **–** Subtraction
5. Bit AND: **&**
6. Bit OR: **|**; Bit XOR: **^**
7. Logical AND: **&&**
8. Logical OR: **||**
9. Comparison operators:
   - **==** equal (double =)
   - **!=** not equal
   - **>** greater than
   - **<** less than
   - **>=** greater than or equal
   - **<=** less than or equal

5.4 Program Directives

■ **ORG:** Set program absolute address

ORG <expression>

Example: `ORG 0x100`

■ **EQU** or **==:** (Double =); define a constant in assembler time

<label> EQU <expression> **<label> = = <expression>**

Example:
```
R0    EQU   0x0
R10   ==    0x10
```

■ **Line Comment:** Use semicolon (;)

; characters

Example: `; This is a line comment`

■ **Block Comment:** Use /* and */

/* characters */

Example:
```
/* This is a multi-line
block comment */
```

■ **EOP:** End of program ROM page

EOP

Examples:
```
ORG     0x20
MOV     A, @0
MOV     0x10, A
EOP
CLR     0x10
```

Above program source will be allocated as follows:

Address	Source Program
-	ORG 0x20
0x0020	MOV A, @0
0x0021	MOV 0x10, A
-	EOP
0x0400	CLR 0x10

■ **END:** End of the program

```
END
```

All codes after the END directive will be discarded.

■ **PROC**, **ENDP:** Procedure definition keywords

```
<label> PROC
        <statements>
        ENDP
```

These keywords are used for defining subroutines and to improve programs' legibility.

Examples**:** Bank0:

```
PROC
BC      0x04, 6
BC      0x04, 7
RET
ENDP
```

NOTE

The PROC/ENDP directives will not take effect while assembling. They only improve the legibility of the program. User has to use the "RET" instruction for subroutine return.

■ **INCLUDE:** Include source files that contain symbol definition, macro, procedure definition, etc. Two types of include statement are available, one applies to **System Default** files, and the other is applicable to **User Defined** files.

The **Include System Default** files are located in the "\include" sub-directory of the IDE directory.

```
INCLUDE <file name>
```

The **Include User Defined** files may or may not have its full path specified. If you do not specify the full path, the default path will be used, that is, the location of the current source files.

```
INCLUDE "file path + file name"
INCLUDE "file name"
```

■ **PUBLIC:** The PUBLIC directive exports symbols to other modules

```
PUBLIC <label> [,<label>]
```

■ **EXTERN:** The directive used to import external symbols from other modules

```
EXTERN <label>[,<label>]
```

■ **VAR:** Defines assembler time variables

```
Label VAR <expression>
```

The value of the variable is valid only during assembly time

■ **MACRO**, **ENDM:** Macro definition statement

```
<label> MACRO <parameters>
        statements
        ENDM
```

Example:

```
Bank0   MACRO
        BC      0x04, 6
        BC      0x04, 7
$MP_Note:                       ; see note below
        MOV     A, NoteBuf
        XOR     A, @CPAUSE
        JBS     SF, Z
        JMP     $MP_Note
        ADD     A, NoteOn
        ENDM
```

NOTE

*Label identifying a Macro element must be prefixed with a dollar sign ($) as shown in the case of "**$MP_Note:**" per above example).*

■ **MACEXIT:** Macro exit statement

MACEXIT

The assembler will not assemble the statements subsequent to the MACEXIT directive

Examples:
```
ABC     VAR     10
Bank0   MACRO
        BC      0x04, 6
        IF ABC>4
                MACEXIT
        ENDIF
        BC      0x04, 7
        ENDM
```

Only the source line "BC 0x04, 6" will be assembled. The remaining lines are ignored (not assembled).

■ **MESSAGE:** User's output message in the assembler time output window

MESSAGE "<characters>"

Examples:
```
ORG     0x0000
MESSAGE "Now set bank to 0."
BC      0x04, 6
BC      0x04, 7
```

The output window will display:

```
"USER MESSAGE: Now set bank to 0"
```

■ **$:** Current program counter value

$

User can use this directive as operand

Examples: JMP $

Jump to current line which is an infinite idle loop.

5.5 Conditional Assembly

■ **IF, ELSEIF and ENDIF:** IF conditional directives

```
IF        <expression>
          <statements>
[ELSEIF   <expression>
          <statements>]
[ELSE
          <statements>]
ENDIF
```

If the expression after the IF directive is "true," the assembler will assemble the subsequent source lines until ELSEIF/ELSE/ENDIF directives are encountered.

Example:

```
ORG   0x0000
Bank  MACRO  num
      IF num==0
      BC     0x04, 6
      BC     0x04, 7
      ELSEIF num==1
      BS     0x04, 6
      BC     0x04, 7
      ELSEIF num==2
      BC     0x04, 6
      BS     0x04, 7
      ELSEIF num==3
      BS     0x04, 6
      BS     0x04, 7
      ELSE
      MESSAGE "Error: Invalid bank number!"
      ENDIF
      ENDM
```

■ **IFE, ELSEIFE** and **ENDIF:** IFE conditional directives

```
IFE         <expression>
            <statements>
[ELSEIFE    <expression>
            <statements>]
[ELSE
            <statements>]
ENDIF
```

If the expression after the IFE directive is "false," the assembler will assemble the subsequent source lines until ELSEIFE/ELSE/ENDIF directives are encountered.

■ **IFDEF, ELSEIFDEF** and **ENDIF:** IFDEF conditional directives

```
IFDEF       <label>
            <statements>
[ELSEIFDEF  <label>
            <statements>]
[ELSE
            <statements>]
ENDIF
```

If the label after the IFDEF directive has been defined, the assembler will assemble the subsequent source lines until ELSEIFDEF/ELSE/ENDIF directives are encountered

Example:
```
ORG     0x0000
TEST    ==  1
IFDEF   TEST
MOV     A, @0
MOV     0x10, A
ENDIF
```

■ **IFNDEF, ELSEIFNDEF** and **ENDIF:** IFNDEF conditional directives

```
IFNDEF      <label>
            <statements>
[ELSEIFNDEF <label>
            <statements>]
[ELSE
            <statements>]
ENDIF
```

If the label after the IFNDEF directive has been defined, the assembler will assemble the subsequent source lines until ELSEIFNDEF/ELSE/ENDIF directives are encountered

5.6 Reserved Words

5.6.1 Directives, Operators

```
+       -       *       /       ==      !=
$       @       (       )       !       ~       %
<<      >>      &       |       ^       &&      ||
<       <=      >       >=

ELSE        ELSIF       ELSEIFDFF   ELSEIFE     ELSEIFNDEF
ENDIF       END         ENDM        ENDMOD      ENDP
EQU         EXTERN      IF          IFE         IFDEF
IFNDEF      INCLUDE     MACRO       MACEXIT     MODULE
ORG         PROC        PUBLIC
```

5.6.2 Instruction Mnemonics

```
ADD     AND     BC      BS      CALL    CLR
COM     COMA    DAA     DEC     DECA    DJZ
DJZA    GET     INC     INCA    IOR     IOW
JBC     JBS     JMP     JZ      JZA     MOV
MPY     NOP     OR      PAGE    RET     RETI
RETL    RETS    RLC     RLCA    RRC     RRCA
SLEP    SUB     SWAP    SWAPA   WDTC    XOR
```

5.6.3 Register Name

```
A
```

5.7 Assembler/Linker Error Messages

5.7.1 Assembler Error Messages

■ **Error A001**

error A001: Can not find the [filename] file

Cause**:** The file is not found in the folder.

Solution**:** Check whether the filename is in error or whether the file exits in the corresponding folder.

■ **Error A002**

error A002: Main and subroutine program can not define [label_name] local label !!

Cause**:** The local label (prefixed with label symbol "$") is incorrectly defined in the main and subroutine program.

Solution**:** Remove the local label from the main or subroutine program.

■ **Error A003**

error A003: The syntax form should be: operation [operand][,operand]

Cause**:** The syntax format of the statements is in error.

Solution**:** Modify format as "operation [operand][,operand]."

■ **Error A004**

error A004: The label [label_name] is redefined

Cause**:** The defined label already exists.

Solution**:** Redefine the label name to make it unique from others.

■ **Error A005**

error A005: The EQU syntax is: label EQU operand

Cause**:** The EQU syntax is in error.

Solution**:** Modify the line to with appropriate syntax.

■ **Error A006**

error A006: The INCLUDE nest depth is over 256

Cause**:** The maximum “INCLUDE” nest depth level is exceeded.

Solution**:** Decrease the nest depth to within the 256 levels.

■ **Error A007**

error A007: The IF conditional expression is in error

Cause**:** The expression “IF conditional” is in error.

Solution**:** Modify the “IF conditional” with correct expression.

■ **Error A008**

error A008: Attempt to divide by zero

Cause**:** An attempt is made to divide the number of expression by zero.

Solution**:** Avoid dividing the expression by zero.

■ **Error A009**

error A009: The assembler dose not support floating point

Cause**:** A non-supported floating point arithmetic is defined as the assembler number system.

Solution**:** Change the floating point to an integer number.

■ **Error A010**

error A010: The symbol [symbol_name] is not defined

Cause**:** The pertinent symbol is not found (undefined).

Solution**:** Define the proper symbol.

■ **Error A011**

error A011: The macro name [macro_name] is redefined

Cause**:** The macro name already exists.

Solution**:** Change the macro name to make it unique from existing ones.

■ **Error A012**

error A012: The parameters name [parameter_name] is the same as label

Cause**:** The parameter name and label are identical.

Solution**:** Redefine the parameter name to differ with label.

■ **Error A013**

error A013: The parameter name [parameter name] is the same as another one

Cause**:** The "parameter name" already exists.

Solution**:** Redefine the macro definition parameter with unique parameter name.

■ **Error A014**

error A014: The number of actual parameter dose not match with formal parameter

Cause**:** The ordinal number of actual parameter does not match with the number in the formal parameter (or vise versa).

Solution**:** Change the ordinal number of either actual or formal parameter to make them match with each other.

■ **Error A015**

error A015: The parameter number [k] does not exist

Cause**:** The ordinal number in the actual parameter is incomplete, i.e., not defined.

Solution**:** Provide the missing ordinal

number or change it with the correct number.

■ **Error A016**

error A016: The external symbol [symbol_name] is the same as defined label

Cause**:** The external symbol name and internal defined label are identical.

Solution**:** Modify the internal defined label to differ with external symbol name.

■ **Error A017**

error A017: Address of ORG is error

Cause**:** The ORG address definition is incorrect.

Solution**:** Redefine the ORG definition with correct address.

■ **Error A018**

error A018: MACEXIT can not be set outside macro

Cause**:** The “MACEXIT” instruction is set outside the macro definition.

Solution**:** Either remove the “MACEXIT” macro instruction or insert it inside the macro definition.

■ **Error A019**

error A019: Parameter must be string variable

Cause**:** The formal parameter is not defined as a string.

Solution**:** Change the formal parameter to string.

■ **Error A020**

error A020: Memory can not be allocated

Cause**:** OS memory saturated.

Solution**:** Close unused applications or any editor window that is currently not used.

■ **Error A021**

error A021: The source statements exceed [k] lines

Cause**:** The number of lines in the source file exceeds the system default limitation.

Solution**:** Split the program into two or more smaller programs.

■ **Error A022**

error A022: The tree is error because of parser error

Cause**:** The program line syntax is in error.

Solution**:** Rewrite and correct the program line.

■ **Error A023**

error A023: [allocated_memory_type] memory fault when memory is allocated

Cause**:** OS memory already full.

Solution**:** Close unused applications or any editor window that is currently not used.

■ **Error A024**

error A024: Set assembler variable must be integer value

Cause**:** The result of expression to the right of "SET" instruction is not an integer value.

Solution**:** Modify the expression into an integer value.

■ **Error A025**

error A025: Address of ORG must be integer value

Cause**:** The ORG address is not an integer value.

Solution**:** Rewrite the ORG address expression into an integer value.

■ **Error A026**

error A026: The [k] address of PC is out of [k] ROM size

Cause**:** The program counter address exceeds the program ROM size.

Solution**:** Decrease the code size.

■ **Error A027**

error A027: The assembler is exceed max pass [k]

Cause**:** The assembler pass exceeds the maximum default pass.

Solution**:** Check the assembler for errors.

■ **Error A028**

error A028: The operand [k] value dose not include the valid data

Cause**:** The operand value is invalid.

Solution**:** Change the operand with valid value.

■ **Error A029**

error A029: The [k] address is conflicted

Cause**:** The program address is in conflict with another.

Solution**:** Use the [Edit]→[Find] menu to find the conflicting address in the list file, then modify the program position with proper address.

■ **Error A030**

error A030: The file [filename] could not be opened

Cause**:** The prescribed filename is in error or missing.

Solution**:** Check filename for entry error or whether the file exists or not.

■ **Error A031**

error A031: The configure file read error

Cause**:** The file configuration format is not correct.

Solution**:** Set up the EM61IDE software again.

■ **Error A032**

error A032: The file [filename] can not be opened

Cause**:** The prescribed filename is in error or missing.

Solution**:** Check filename for entry error or whether it exists or not.

■ **Error A033**

error A033: The macro is not defined

Cause**:** The stated macro name cannot be found, i.e., not defined or in error.

Solution**:** Define a new macro or correct the existing macro name.

■ **Error A034**

error A034: The expression can not be calculated

Cause**:** The expression format is in error.

Solution**:** Modify the expression with correct format.

■ **Error A035**

error A035: The operation [operation_name] is not defined

Cause**:** The operation name is not found or not defined by EM61IDE.

Solution**:** Check the EM61IDE user menu for valid operators.

■ **Error A036**

error A036: The PUBLIC or EXTERN label [label_name] must be address label

Cause**:** The label defined by “PUBLIC” or “EXTERN” is not an address label.

Solution**:** Remove the public or extern variable.

■ **Error A037**

error A037: The operand value can not be calculated

Cause**:** The operand expression format is incorrect.

Solution**:** Modify the operand expression with correct format.

■ **Error A038**

error A038: The symbol [symbol_name] is not extern symbol

Cause**:** The symbol is not defined as internal nor external.

Solution**:** Redefine symbol with appropriate symbol name.

■ **Error A039**

error A039: The reference number is over [k] limits

Cause**:** The reference number of “PUBLIC” or “EXTERN” exceeds limitation.

Solution**:** Decrease the number of external symbols.

■ **Error A040**

error A040: The length of file name [file_name] exceeds 256

Cause**:** The filename length is over 256 characters.

Solution**:** Decrease length of filename to within 256 characters.

5.7.2 Linker Error Messages

■ **Error L001**

error L001: Memory of stack overflow

Cause**:** OS unable to allocate extra memory (memory full).

Solution**:** Close unused applications or any editor window that is currently not used.

■ **Error L002**

error L002: The [file_name] not found

Cause**:** The defined filename is missing or cannot be identified.

Solution**:** Check filename for entry error or whether it exists or not.

■ **Error L003**

error L003: The Object file format dose not belong to EMC object file format

Cause**:** The object file format is not compatible to EMC object file format.

Solution**:** Re-assemble the source file with EM61IDE.

■ **Error L004**

error L004: Symbol [symbol_name] is not defined

Cause**:** The symbol is missing, i.e., not defined.

Solution**:** Define the symbol.

■ **Error L005**

error L005: Public symbol [symbol_name] is conflicted

Cause**:** The defined public symbol is in conflict with other symbol.

Solution**:** Redefine the public symbol.

■ **Error L006**

error L006: ROM address [k] is conflicted

Cause**:** The program address is in conflict with another.

Solution**:** Use the [Edit]→[Find] menu to find the conflicting address in the list file, then modify the program position with proper address.

■ **Error L007**

error L007: The file [file_name] can not be created

Cause**:** Disk or system refused to create the file.

Solution**:** Check disk for capacity. If okay, then try rebooting the system.

■ **Error L008**

error L008: Line [k]: The machine address [k] exceeds ROM size [k]

Cause**:** The program address exceeds maximum ROM size.

Solution**:** Decrease the program code.

■ **Error L009**

error L009: No project file active

Cause**:** No project is opened nor created.

Solution**:** Open or create a project.

■ **Error L010**

error L010: No output window found

Cause**:** Output window is not displayed.

Solution**:** Click on [View]→[Output]. The Output window should now display.

Chapter 6
Easy Program

6.1 Writing an EASY Format® (ESY) Program

6.1.1 File Format

```
[InputState]
      …
      …
[OutputState]
      …
      …
[Path]
Path0:
      …
      …
[Background]
      …
      …
[Asm]
      …
      …
[TCCAsm]
      …
      …
[Macro]
      …
      …
[Symbol]
      …
      …
```

The EASY program does not support multifile. An EASY format source file consisted of several blocks, i.e., [InputState], [OutputState], [Path], etc. The blocks can be written in different order and all words are case-insensitive.

6.1.1.1 [InputState]:

```
StateName : PathName1, PathName2, …, PathNameN
```

InputState is defined for each input trigger state assignment. You can define up to 256 states with different input state names. The state name must be followed by the 'N' trigger of selection options, where N is the option selected in the project for EASY-N setting.

The trigger selection options can be one of the following:

p_name when the corresponding trigger is invoked (or pressed in case of keypad), the path equation with path name "p_name" is executed.

/p_name when the corresponding trigger is released, the path equation with path name "p_name" is executed.

p_name1/p_name2 when the corresponding trigger is invoked (or pressed in case of keypad), the path equation with path name "p_name1" is executed. When the corresponding trigger is released, the path equation with path name "p_name2" is executed.

X disable the corresponding trigger (no action).

6.1.1.2 [OutputState]:

```
OutputName : Pn=[O7  O6  O5  O4  O3   O2  O1  O0]
```

"Pn" stands for Port 2, 3, 4, and 5, i.e., P2 means Port 2 and P3 means Port 3, etc. The output symbol "O0," "O1," "O2," etc., stand for Pin 0, Pin 1 and Pin 2, etc., respectively. Each output pin status may be specified with one of the following output symbols**:**

1 set the corresponding I/O pin as "H"

0 set the corresponding I/O pin as "L"

P+ send a pulse train beginning from "H" to the corresponding I/O pin

P– send a pulse train beginning from "L" to the corresponding I/O pin

S+ send a positive pulse to the corresponding I/O pin

S– send a negative pulse to the corresponding I/O pin

F set the designated I/O pins as tri-state

X keep the current state of the corresponding output pin unchanged or used as "not used" pin

6.1.1.3 [Path]:

PathName : parameter0, parameter1,…,parameterN

Users can define as many path equations as needed and assign each with different path names, i.e., p_name1, p_name2, etc. The path name "path0" must exist in an EASY program as the starting point of the program. Each path equation line should contain less than 255 characters. If a line is inadequate to define a path equation, additional lines may be added. But the additional lines must begin with an ampersand (&) to indicate that it is a continuation to the previous line of the same path equation definition.

Refer to Section 6.3 *Path Equation Parameters and Definitions* for the list of parameters and its corresponding definitions.

6.1.1.4 [Background]:

BackgroundName : parameter0, parameter1, …, parameterN

Background allows the program to execute 2 commands simultaneously. The major flow of a program must defined in [Path] section. Background is applicable only when the program needs to perform two things at the same time. The switching between main path and background path occurs when **delay** or **voice/melody playback** command is executed. For example, play voice in the main path and perform output in background, or play voice in background and perform time out in the main path. Background process is enabled in the main path by BK_ENABLE(name) command and can be disabled by BK_DISABLE. All parameters in [Path] section can be used in [Background] section except BK_ENABLE and BK_DISABLE. The path name in some parameters can only be referenced to the name of background path name.

6.1.1.5 [Asm]: In-EASY Assembly

```
[ASM]
  AsmName1 {
  <statements>
  …
  label:
  <statements>
  …
  }

  AsmName2 {
  <statements>
  …
  }
```

In-EASY assembly provides a block for users to write assembly code in EASY format. The syntax of In-EASY assembly is the same as the assembly language described in Chapter 5, but without assembly directives. In-EASY assembly can use local label in its definition block.

The In-EASY assembly definition starts from an open brace symbol ({) and ends with a close brace (}). The symbol is used in Path or Background section to execute the assembly block. The following instructions should not be used in the In-EASY assembly to avoid influencing the EASY Format internal routine**:**

```
"MOV 0x02, A" : change TCC timer value
"MOV 0x07, A" : change TCC2 timer reload value
"CALL"
"RET"
"SLEP"
```

Refer to Section 6.6, *Guidelines to Writing Assembly Code* for more helpful hints.

6.1.1.6 [TCCAsm]:

TCCAsm block provides an entry point for user defined routine to be automatically executed at every TCC timer overflow. This is a special arrangement within the In-EASY assembly whereby you need not bother to take any action at every timer overflow. The time interval is specified by the "TCC Resolution Option". Only one piece of assembly code is written in this block. Unlike the format used under In-EASY assembly block, the "AsmName", "{." and "}" are not required for block definition. The programming rule is the same as "[Asm]" section.

Refer to Section 6.6, *Guidelines to Writing Assembly Code* for more helpful hints.

6.1.1.7 [TCC2Asm]

TCC2Asm block provides an entry point for user defined routine to be automatically executed at every TCC2 timer underflow. Only the EM61 series have this function. This is a special arrangement within the In-EASY assembly whereby you need not bother to take any action at every timer overflow. The time interval is specified by the "TCC2 Resolution Option." Only one piece of assembly code is written in this block. Unlike the format used under In-EASY assembly block, "AsmName", "{" and "}" are not required for block definition. The programming rule is the same as "[Asm]" section.

6.1.1.8 [Macro]:

```
MacroName: parameter0 parameter1 … parameterN
```

The macro section specifies the macro equations.

6.1.1.9 [Symbol]:

```
SymbolName == data
SymbolName == Mi
SymbolName EQU data
SymbolName EQU Mi
```

The symbol section defines the symbol or "names" for data registers or constant data.

Example:

Define registers M0 as "X," M1 as "RESULT," and symbol "TRUE" as 1.

```
X == M0
RESULT EQU M1
TRUE == 1
```

6.2 Applicable Registers

The registers are the data RAM. Each register is 8-bit in width and has the following format**.**

`Mi` (where "*i*" is the valid register number 0~44 shown in the table below.)

The registers are useful in storing data or in performing arithmetic/logical/ conditional branch operation. You can assign symbol "names" for these registers in the [Symbol] section (see Section 6.1.1.8 above) which could help you easily remember the particular registers.

The valid registers for EM61000 series are as follows**:**

Body	Valid Registers
EM61001	M0~M110
EM61002	M0~M110
EM61100	M0~ M101
EM61200	M0~M101
EM61300	M0~ M101
EM61400	M0~ M101
EM61500	M0~ M101

6.3 Path Equation Parameters and Definitions

Each path equation definition within a [Path] block may contain one or more parameters as may be required as shown in the table below.

The following explains some of the unique segments used in the parameters**:**

Mi	Registers as defined in Section 6.2.
Speech Section Name	The speech waveform file (*.wav) inserted in the IDE project.
Melody Section Name	The melody file inserted in the IDE project. It could be the MIDI (*.mid) or Elan's melody intermediary file (*.mit).
Head Waveform Name	The head waveform file inserted in the IDE project.
Tail Waveform Name	The tail waveform file inserted in the IDE project.
Data Waveform Name	The data file inserted in the IDE project.

Parameter	Definition
Path Name	Jump to the other path as defined in the [Path] block.
Asm Name	Call the In-EASY assembly block defined in the [Asm] block.
DELAY(n)	Delay "n" seconds. The minimum delay unit is 0.001 seconds. For example, Delay(0.2) means 0.2 second delay and Delay(8.6) means 8.6 seconds delay.
RATE=n	Define the flash rate of flash output pins that are set to "P+" or "P–". "n" can be any number from 1 to 8 of the eight flash rate variables 16/n Hz.

RandN?=data/Mj:p_name	If the contents of register “RandN” **equal** to data/register “Mj” contents, the path equation with the path name “p_name” will be executed. “N” can be any number from 1 to 3.
RandN?<>data/Mj:p_name	If the contents of register “RandN” **do not equal** to data/register “Mj” contents, the path equation with the path name “p_name” will be executed. “N” can be any number from 1 to 3.
RandN?!=data/Mj:p_name	If the contents of register “RandN” **do not equal** to data/register “Mj” contents, the path equation with the path name “p_name” will be executed. “N” can be any number from 1 to 3.
RandN?>data/Mj:p_name	If the contents of register “RandN” are **greater than** data/register “Mj” contents, the path equation with the path name “p_name” will be executed. “N” can be any number from 1 to 3.
RandN?<data/Mj:p_name	If the contents of register “RandN” are **less than** data/register “Mj” contents, the path equation with the path name “p_name” will be executed. “N” can be any number from 1 to 3.
RandN?>=data/Mj:p_name	If the contents of register “RandN” are **greater than** or **equal** to data/register “Mj” contents, the path equation with the path name “p_name” will be executed. “N” can be any number from 1 to 3.
RandN?<=data/Mj:p_name	If the contents of register “RandN” are **less than** or **equal** to data/register “Mj” contents, the path equation with the path name “p_name” will be executed. “N” can be any number from 1 to 3.
RandN:[p_0 x p_2 …]	If the contents of register “RandN” are **equal to “0,”** the path equation with path name “p_0” will be executed. If the contents of register “RandN” are **equal to “1**,” and the corresponding path encounters “X” (don’t care), the equation will execute the next parameter. If the contents of register “RandN” are **equal to “2,”** the path equation with path name “p_2” will be executed, so on and so forth. The maximum number of path names that can be accommodated in the bracket”[ ]” is 128. “N” can be any number from 1 to 3.
Mi=data/Mj	Define the contents of register “Mi” as data or that of register “Mj,” where data can be any value from 0 to 255.
Mi=RC_ROTATE(Mj)	Define the contents of register “Mi” as the result of **right rotation** of the contents of register “Mj” with carry flag. The carry flag will be changed.
Mi=LC_ROTATE(Mj)	Define the contents of register “Mi” as the result of **left rotation** of the contents of register “Mj” with carry flag. The carry flag will be changed.
Mi=R_ROTATE(Mj)	Define the contents of register “Mi” as the result of **right rotation** of the contents of register “Mj.” The carry flag will be changed.
Mi=L_ROTATE(Mj)	Define the contents of register “Mi” as the result of **left rotation** of the contents of register “Mj.” The carry flag will be changed.

Mi=R_SHIFT(Mj)	Define the contents of register “Mi” as the result of one bit **right shift** of the contents of register “Mj”. The MSB in register “Mi” becomes “0” The carry flag will be changed.
Mi=L_SHIFT(Mj)	Define the contents of register “Mi” as the result of one bit **left shift** of the contents of register “Mj”. The LSB in register “Mi” will be “0”. The carry flag will be changed.
Mi=AR_SHIFT(Mj)	Define the contents of register “Mi” as the result of one bit arithmetic **right shift** of the contents of register “Mj”. The MSB in register “Mi” will be the sign bit of register “Mj”. The carry flag will be changed.
Mi=P1	Set the contents of Mi as the contents that are read from **Port 1**.
Mi=P2	Set the contents of Mi as the contents that is read from **Port 2**.
Mi=P3	SET the contents of Mi as the contents that is read from **Port 3**.
Mi=P4	Set the contents of Mi as the contents that are read from **Port 4**. **NOTE** *Not applicable to EM61001/002.*
Mi=P5	Set the contents of Mi as the contents that are read from **Port 5**. **NOTE** *Not applicable to EM61001/ 002/100/200/300.*
Mi=Mj+data/Mj-data	Define the contents of register “Mi” as the contents of register “Mj” plus/subtract **data**.
Mi=Mj+Mk/Mj-Mk	Define the contents of register “Mi” as the contents of register “Mj” plus/subtract the contents of register “**Mk**.”
Mi,Mj=Mk*data	Define the contents of registers “Mi” & “Mj” as the contents of register “Mk” multiplied by **data** (signed multiplication). The high byte of the result will be stored in “Mi,” the low byte will be stored in “Mj.”
Mi,Mj=Mk*Ml	Define the contents of register “Mi” & “Mj” as the contents of register “Mk” multiplied by “**Ml**” (signed multiplication). The high byte of the result will be stored in “Mi,” the low byte will be stored in “Mj.”
Mi=Mk/data	Define the contents of register “Mi” as the resulting contents of register “Mk” **divided by data** (signed division).
Mi=Mk/Ml	Define the contents of register “Mi” as the resulting contents of register “Mk” **divided by the contents of register “Ml”** (signed division).
Mi=Mk%data	Define the contents of register “Mi” as reminder of the contents of register “Mk” **divided by data** (signed division).
Mi=Mk%Ml	Define the contents of register “Mi” as reminder of the contents of register “Mk” **divided by the contents of register “Ml”** (signed division).

Mi=Mj.and.data/Mj.and.Mk	Define the contents of register “Mi” as the contents of register “Mj” **AND** data/that of register “Mk.”
Mi=Mj.or.data/Mj. and.Mk	Define the contents of register “Mi” as the contents of register “Mj” **OR** data/that of register “Mk.”
Mi=Mj.xor.data/Mj.and.Mk	Define the contents of register “Mi” as the contents of register “Mj” **XOR** data/that of register “Mk.”
Mi=.inv.Mj	Define the contents of register “Mi” as the result of INVERTing that of register “Mj.”
Mi(j)=1/0	Define the “jth” bit of register “Mi” as 1/0, where ‘j’ can be any value from 0 to 7.
Mi(j)=~Mk(l)	Define the “jth” bit of register “Mi” as the toggled value of “lth” bit of register “Mk”, where ‘j’ and ‘l’ can be any value from 0 to 7.
Mi(j)=Pn.x	Define the “jth” bit of register “Mi” as Pn.x. The pin specified in “Pn.x” can be either input or output. ‘j’ can be any value from 0 to 7.
P2=Mi/data	Define the output contents of the **Port 2** as the content of register “Mi” or data. Only the pin configured as output will be set.
P3=Mi/data	Define the output contents of **Port 3** as the content of Register “Mi” or data. Only the pin configured as output will be set.
P4=Mi/data	Define the output contents of **Port 4** as the content of Register “Mi” or data. Only the pin configured as output will be set. ***NOTE*** *Not applicable to EM61001/002*
P5=Mi/data	Define the output contents of **Port 5** as the content of Register “Mi” or data. Only the pin configured as output will be set. **NOTE** *Not applicable to EM61001/ 002/100/200/300.*
Mi?=data:p_name	If the contents of register “Mi” **equal to data**, the path equation with the path name “p_name” will be executed
Mi?=Mj:p_name	If the contents of register “Mi” **equal to that of register “Mj,”** the path equation with the path name “p_name” will be executed
Mi?<>data:p_name	If the contents of register “Mi” **do not equal to data**, the path equation with the path name “p_name” will be executed
Mi?<>Mj:p_name	If the contents of register “Mi” **do not equal to that of register “Mj,”** the path equation with the path name “p_name” will be executed
Mi?!=data:p_name	If the contents of register “Mi” **do not equal to data**, the path equation with the path name “p_name” will be executed

Mi?!=Mj:p_name	If the contents of register “Mi” **do not equal** to that of register “Mj,” the path equation with the path name “p_name” will be executed.
Mi?>data:p_name	If the contents of register “Mi” are **greater than data**, the path equation with the path name “p_name” will be executed.
Mi?>Mj:p_name	If the contents of register “Mi” are **greater than register** “Mj,” the path equation with the path name “p_name” will be executed.
Mi?<data:p_name	If the contents of register “Mi” are **less than data**, the path equation with the path name “p_name” will be executed.
Mi?<Mj:p_name	If the contents of register “Mi” are **less than that of register “Mj,”** the path equation with the path name “p_name” will be executed.
Mi?>=data:p_name	If the contents of register “Mi” are **greater than or equal to data**, the path equation with the path name “p_name” will be executed.
Mi?>=Mj:p_name	If the contents of register “Mi” are **greater than or equal to that of register “Mj,”** the path equation with the path name “p_name” will be executed.
Mi?<=data:p_name	If the contents of register “Mi” are **less than or equal to data**, the path equation with the path name “p_name” will be executed.
Mi?<=Mj:p_name	If the contents of register “Mi” are **less than or equal to that of register “Mj,”** the path equation with the path name “p_name” will be executed.
Mi(j)?=1/0:p_name	if the “jth” bit of register “Mi” are equal to 1/0, the path equation with the path name “p_name” will be executed.
Mi:[p_0 x p_2 …]	if the contents of register “Mi” are **equal to “0,”** the path equation with the path name “p_0” will be executed. If the contents of register “Mi” are **equal to “1,”** and the corresponding path encounters “X” (don’t care), the equation will execute the next parameter. If the contents of register “Mi” are **equal to “2,”** the path equation with path name “p_2” will be executed, so on and so forth. The maximum number of path names that can be accommodated in the bracket ”[ ]” is 128.
Mi:[k0:p_0 k1:p_1…]	If the contents of register “Mi” **equal to “k0,”** the path equation with path name “p_0” will be executed. If the contents of register “Mi” **equal to “k1,”** the path equation with path name “p_1” the equation will be executed and so on and so forth. The maximum number of path names that can be accommodated in the bracket”[ ]” is 128.
CARRY?:p_name	If the carry flag is “1,” the path equation with the pathname “p_name” will be executed.

ZERO?:p_name	If the zero flag is “1”, the path equation with pathname “p_name” will be executed.
TRn?=H/L:p_name	If the trigger TRn is “H”/”L,” the path equation with path name “p_name” will be executed. ‘n’ can be from 1 and up to the number N of EASY-N.

NOTE
This parameter will not work properly when customized keyboard scan routine is used.

{param0 param1 …}*N	Execute a set of parameters N times. N=0…255.
{param0 param1 …}*Mi	Execute a set of parameters repeatedly. The execution time will be in accordance with the contents of register “Mi.”
Mi=VROMDATA	Define the contents of register “Mi” as the data to be pointed by data pointer of VROM for table-read. The data pointer will point to the next data automatically.
DP=Data Section Name	Assign the address of data pointer as “Data Section Name” for voice ROM.
DP_L=data/Mi	Assign the **low byte** address of data pointer for voice ROM.
DP_M=data/Mi	Assign the **medium byte** address of data pointer for voice ROM.
DP_H=data/Mi	Assign the **high byte** address of data pointer for voice ROM.
Mi=DP_L	Define the contents of register “Mi” as the **low byte** address of data pointer for voice ROM.
Mi=DP_M	Define the contents of register “Mi” as the **medium byte** address of data pointer for voice ROM.
Mi=DP_H	Define the contents of register “Mi” as the **high byte** address of data pointer for voice ROM.
DP=DP+data/Mi	Add the data pointer with data or contents of register “Mi.”
TCC2_ON	Turn on TCC2 timer interrupt.
TCC2_OFF	Turn off TCC2 timer interrupt.
MDY:Melody Section Name	Play the melody file. It **will call** the scan loop and execute the next parameter till the playing stops.
[MDY:Melody Section Name]	Play the melody file **without calling** the scan loop nor executing the next parameter immediately.
MDY:Melody Section Name*N	Play the melody file and repeat N times.
[MDY:Melody Section Name]*N	Play the melody file and repeat N times **without calling** the scan loop.

MnTONE/i=inst_data/f=freq_ data/w=wav_data/e=env_data	Play the tone file in the melody Channel n. Specify the frequency index and initial envelope. Where**:** "**n**" =1…4 (for EM61001/002) "**n**" =1…8 (for EM61100…500) "**i**" = 0… max. instrument number-1 "**inst_data**" is the instrument number "**freq_data**" is the frequency index (=0…26) "**w_data**" is the waveform index of the instrument (= 0…2) "**env_data**" is the envelope (= 0…127)
MCHn:STOP	Stop the tone waveform playback in melody Channel n, where**:** "**n**" =1…4 (for EM61001/002) "**n**" =1…8 (for EM61100…500)
MPLAY?:p_name	If a melody file is playing, the path equation with path name "p_name" will be executed
MORGINST=data/Mi	Set which **original instrument** ("data" or contents of register "Mi") is to be replaced.
MNEWINST=data/Mi	Set which **new instrument** ("data" or content of register "Mi") will replace the original instrument.
MXINST	Change the original instrument (set by MORGINST) to new instrument (set by MNEWINST). Before using this parameter, set the original/new instrument with "MORGINST" and "MNEWINST" parameters beforehand.
MnINST=data/Mi	Set the **instrument** of melody Channel n to data or to the contents of the register "Mi" during melody playback, where**:** "**n**" =1…4 (for EM61001/002) "**n**" =1…8 (for EM61100…500)
MnENV=data/Mi	Define the **envelope** of melody Channel n as data or contents of the register "Mi" during melody playback, where**:** "**n**" =1…4 (for EM61001/002) "**n**" =1…8 (for EM61100…500)
MnVOL=data/Mi	Define the **volume** of melody Channel n as data or as contents of the register "Mi" during melody playback, where**:** "**n**" =1…4 (for EM61001/002) "**n**" =1…8 (for EM61100…500)
MnNOTEON?:p_name	Activate execution of the path equation with path name "p_name" when the melody note of Channel n is ON. The default value is "OFF."
MPAUSE	Pause the current melody playback. This is a toggle function.
MMUTE	Mute the current melody playback. This is a toggle function.
TEMPO=data	Set tempo value. The valid data is 40 to 240 beat/min.
TEMPO++	Increase tempo value by one step.
TEMPO--	Decrease tempo value by one step.

MSTOP	Stop the current melody playback.
DAC1/ON_OFF/RANGE =data1/VOL=data2/ASSIGN =chan_name1+ …channel_name2+…	Set up **DAC1** control. The arguments “ON_OFF,” “RANGE,” “VOL,” and “ASSIGN” are optional. Where**:** “ON_OFF = **ON**” turns on DAC “ON_OFF = **OFF**” turns off DAC “**RANGE = data1 or Mi**” is the value 0…2 “**VOL = data or Mj**” is the value 0…7 “**ASSIGN = channel_name+…**” assigns which channel is to be sent into DAC mixer. The “chan_name“ options are**:** “**n**” =1…4 (for EM61001/002) “**n**” =1…8 (for EM61100…500)
DAC2/ON_OFF/RANGE =data1/VOL=data2/ASSIGN =chan_name1+ …channel_name2+… (M61001/002 not supported)	Set up **DAC2** control. the argumentS “ON_OFF,” “RANGE,” “VOL,” and “ASSIGN” are optional. Where: “ON_OFF = **ON**” turns on DAC “ON_OFF = **OFF**” turns off DAC “**RANGE = data1 or Mi**” value is 0…2 “**VOL = data or Mj**” is the value is 0…7 “**ASSIGN = channel_name+..**” assigns which channel is to be into DAC mixer. The “chan_name“ options are**:** “**n**” =1…8 (for EM61100…500 only)
HEAD:Head Waveform Name	Play a ramp-up waveform specified by “Head Waveform Name” on **DAC1 & 2**, and turn them on.
HEAD1:Head Waveform Name	Play a ramp-up waveform specified by “Head Waveform Name” on **DAC1** and turn it on.
HEAD2:Head Waveform Name (M61001/002 not supported)	Play a ramp-up waveform specified by “Head Waveform Name” on **DAC2** and turn it on.
TAIL:Tail Waveform Name	Play a ramp-down waveform specified by “Tail Waveform Name” on **DAC1 & 2** and turn them off.
TAIL1:Tail Waveform Name	Play a ramp-down waveform specified by “Tail Waveform Name” on **DAC1** and turn it off
TAIL2:Tail Waveform Name (M61001/002 not supported)	Play a ramp-down waveform specified by “Tail Waveform Name” on **DAC2** and turn it off.
BK_ENABLE (BackgroundName)	Enable a background equation defined in [Background] block (see [Background] block in Section 6.1.1.4).
BK_DISABLE	Disable background equation (see [Background] block in Section 6.1.1.4).

END	Power down the chip and stop the program

NOTE

1. *The "Speech Section Name", "Head Waveform Name" and "Tail Waveform Name" are voice filenames (with "*.wav" extension name or simply filename). The file name will not include file its path.*
2. *The "Melody Section Name" is MIDI filename (with "*.mid" extension name or simply filename). The filename will not include its path.*
3. *The "Data Section Name" is binary user data filename (with extension name or simply file name). The filename will not include its path.*
4. *You should add the voice/data file into the project at the beginning of the IDE tool operation. Then apply these voice/data filenames as parameters in the EASY program. While compiling, the IDE tool will provide the voice file information that was included in the project beforehand. The EASY compiler should check whether the files specified in the EASY program, exist or not. If the filename does not exist, it should display an error message.*

6.4 Reserved Words

The following words are reserved for EASY compiler's internal usage and should NOT be used when defining your own generated labels, variable names, macros, and other parameters.

ASM	ASSIGN	AR_SHIFT	BACKGROUND	BK_ENABLE
BK_DISABLE	CARRY	DELAY	END	EQU
F	H	INPUTSTATE	L	L_ROTATE
L_SHIFT	LC_ROTATE	MACRO	OUTPUTSTATE	PATH
RATE	R_ROTATE	R_SHIFT	RC_ROTATE	STOP
SYMBOL	TCCASM	X	ZERO	DP

The following reserved words contain subscripts character which denotes number.

DAC*i* *(DAC1, DAC2)*	**M***i* *(M0 to M44)*	**M***i***TONE** *(M1TONE, M2TONE, M3TONE, M4TONE)*	**P***i* *(P1, P2, P3 P4 P5)*	**RAND***i* *(RAND1, RAND2 RAND3)*	**Tr***i* *(TRI toTR 64)*

6.5 Limitations

The following are some of the existing limitations for the current version of the EASY format program.

Item	Limit	Description
Input state name	≤255	The total number of Input state names
Trigger	≤64	Up to "EASY-64"
Output state name	≤255	The total number of output state names
Background re-entry	≤128*128-1	Background can use: 1) DELAY(n); 2) play midi & speech (without brackets "[…]"); 3) main path execution limited to BK_ENABLE() total value
Repeat times of loop parameter	≤255	"{parameter0 pparameter1 …}*N." See Note below for maximum "N" value
Repeat times of voice-playing	≤255	"VoiceParameter*N" or"[VoiceParameter]*N" See Note for maximum "N" value
DELAY(n)	>= 0.001s	Delay time valid value: EASY routine maximum internal delay time is 63.75s/delay. If the assigned time exceeds the value, the compiler will automatically perform a double-delay which total delay time equals to the defined time.
Name:[p_0 p_1 …]	≤128	"p_128" is the maximum value that can be assigned
Name?[k1:p1 k2:p2 …]	≤256	"k256:p256" is the maximum value that can be used
M*i*	IC body dependent	The maximum number of registers that can be defined depends on the type of IC body you are using. Refer to Section 6.2 "*Applicable Registers*" for details.
Parameter	Program ROM space dependent	The maximum capacity that can be defined for each command depends on the available space of the program ROM in use
Path		
In-EASY assemble		
Macro		
Symbol		

NOTE: Loop parameter can include voice-playing-repeat parameter.

6.6 Guidelines to Writing Assembly Code

When defining blocks [Asm], [TCCAsm] and user-defined scan key option, be sure to take note of the following:

1. The following symbols are valid but cannot be assigned twice.
   - The Data Registers (note that M*i* (*i*= 0, 1, 2,…) is already defined by EASY compiler).
   - EM61000 series register definitions, i.e., *R0, TCC, PC, SF, RSR, INTC, MTC, R7, R8, R9, MODE, PORT1, PORT2, PORT3, MULT, PRODL,*

PRODH, C, DC, Z, P, T, PS0, PS1, PS2, TCIF, MIF, TCIE, MIE, ENI, WTE, WAKEL, WAKEH, P2L_L, P2L_H, P1L, IOC5, IOC6, ADDL, ADDM, ADDH, DATA, RATE, DA1, DA2, ENV, F1, F2, IOCD, IOCE, CSCH1, CSCH2, CMCH1, CMCH2, CMCH3, CMCH4, CVROM, CDAC1, CDAC2, SENT, SENA, MENA, and DAC_ON.

The file "*ezinc\em6xxxx.def"* included in the EM61000 IDE package also lists the register definitions which are located in between symbols *";.memon" ~ ";.memoff"* of the file.

2. The valid macros are *BANK, FDEC, FDECA, FMOV, FJBC, FJBS, FBC, FBS, FJZ, FJZA, FSWAP, FSWAPA, FDJZ, FDJZA, FINC, FINCA, FCLR, FCOM, FCOMA, FRRC, FRRCA, FRLC, FRLCA, FADD, FSUB, FAND, FXOR, FOR, MOV, SUB, OR, AND, XOR, ADD, FMPY, FCALL,* and *FJMP*.

   Refer to "*ezinc\user_emc60.inc"* included in the EM61000 IDE package for the definitions of each item.

3. When writing assembly codes, EASY compiler will generate internal labels prefixed with "*_ELAN_*" except for the following**:**
   - Path name
   - ASM block name

4. The EASY compiler will locate these blocks and the user-defined scan key routine at the end of the end of the program without optimization. EASY cannot guarantee that they will be located in the same page either. Therefore, you have to use **FCALL** and **FJMP** in lieu of **CALL** and **JMP** commands to place them in the same page.

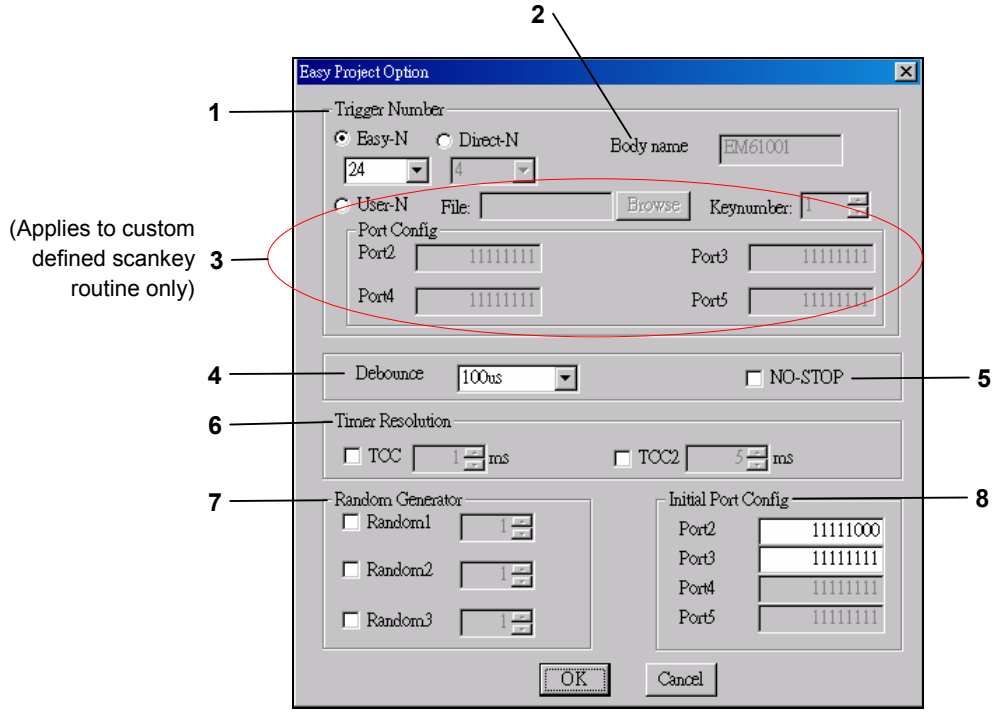
6.7 Initial Condition

After power on or reset, the EM61000 Series IC will enter into its hardware initialized condition (as described in the EM61000 User Manual). Then the EASY program will further implement (through software) additional initial conditions of its own into the chip

1. All I/O ports will be set to output except for the ones that are configured as input in the Trigger option. The output initial value is in accordance with port configuration status defined in EASY Project Option dialog (see figure below).
2. All registers (M0, M1, …) are cleared (set to "0").
3. Speech Channel 1, melody Channels 1 & 2 are assigned to DAC1. Speech Channel 2, melody Channels 3 & 4 are assigned to DAC2.

6.8 Options for Configuring EASY Project

The following paragraphs provide the options required to configure an EASY project (as illustrated in the Easy Project Option dialog shown below). The EASY compiler (under IDE) will read the selected options when compiling the EASY project.



"Easy Project Option" Dialog

1 "Trigger Number" Option

This option specifies the input trigger type for the EASY program. The valid options are: DIRECT-8…16, EASY-24/32/40/48/56/64.

Input Trigger	Port 1 Pin in Use	Port 2 Pin in Use	Port 3 Pin in Use
DIRECT-4	–	0…3	–
DIRECT-8	0…7	–	–
DIRECT-9	0…7	0	–
DIRECT-10	0…7	0, 1	–
DIRECT-11	0…7	0…2	–
DIRECT-12	0…7	0…3	–
DIRECT-13	0…7	0…4	–
DIRECT-14	0…7	0…5	–
DIRECT-15	0…7	0…6	–
DIRECT-16	0…7	0…7	–
EASY-16	–	0…7	–
EASY-24	0…7	0…2	–
EASY-32	0…7	0…3	–
EASY-40	0…7	0…4	–
EASY-48	0…7	0…5	–
EASY-56	0…7	0…6	–
EASY-64	0…7	0…7	–

The available trigger key configuration for each body is as follows**:**

BODY	DIRECT-N	EASY-N
EM61001	4, 8…16	24/32/40/48/56/64
EM61002	4, 8…16	24/32/40/48/56/64
EM61100	4, 8…16	24/32/40/48/56/64
EM61200	4, 8…16	24/32/40/48/56/64
EM61300	4, 8…16	24/32/40/48/56/64
EM61400	4, 8…16	24/32/40/48/56/64
EM61500	4, 8…16	24/32/40/48/56/64

2 Chip "Body Name" Option

This option specifies the particular EM61000 Series chip body to be processed under EASY program. The valid body name is one of the following**:**

- EM61001
- EM61002
- EM61100
- EM61200
- EM61300
- EM61400
- EM61500
- EM61600

The program ROM, voice ROM, and I/O port configurations for each body are as follows**:**

Body	PROM	VROM	Port 1	Port 2	Port 3	Port 4	Port 5	DAC
EM61001	8Kwords	32Kbytes	Input	I/O	I/O	-	-	1
EM61002	8Kwords	32Kbytes	Input	I/O	I/O	I/O	-	1
EM61100	8Kwords	64Kbytes	Input	I/O	I/O	I/O	-	2
EM61200	8Kwords	128Kbytes	Input	I/O	I/O	I/O	-	2
EM61300	8Kwords	256Kbytes	Input	I/O	I/O	I/O		2
EM61400	8Kwords	512Kbytes	Input	I/O	I/O	I/O	Output	2
EM61500	8Kwords	1024Kbytes	Input	I/O	I/O	I/O	Output	2

NOTE: Each input/output port is 8 bits in width.

3 Scankey Routine "User-N" Option

If you want to use a keyboard configuration other than the one provided by EASY compiler, you can write your own keyboard scan routine and enter the file into the dialog **File** box to replace the one from EASY compiler after the **User N** check box is enabled. See Section 6.8.1 for the details on how to write a user scankey routine.

You also need to enter the number of keys of your keyboard into the **Keynumber** box and the output configuration of the corresponding ports into their respective **Port Config** boxes.

4 "Debounce" Option

This box allows you to define the debounce time of keyboard scan. The valid value can be set as 100μs, 20000μs, or 80000μs.

5 "NO-STOP" Option

If you want to allow voice playback continuously when trigger is received, the **NO-STOP** check box should be enabled. If this option is disabled, the voice playback is stopped once trigger is received.

6 "Timer Resolution" Option

This option is used to set the TCC and TCC2 timer interval for TCCAsm and TCC2Asm respectively. The TCC value ranges from 1 to 50 ms (positive integer only) while that of TCC2 ranges from 0.1 to 200 ms. The In-EASY assembly in the TCCAsm and TCC2Asm sections will be accessed according to this time interval.

7 Number of "Random Generator" Option

This option sets up the random number generator. A total of three random generators are offered in the EASY program. This option sets the range of each random number generator as follows**:**
Random1=0…N
Random2=0…N
Random3=0…N

Where N ranges from 1 to 127.

8 "Initial Port Configuration" Option

Set the power-on initial condition for output ports. The setting has no effect on the pins marked as "X" in the tables on *Output Port Configuration for EASY-N and DIRECT-N* for different EASY types. These pins will set as High when automatically powered-on by compiler.

6.8.1 How to Write a Custom Scankey Routine

The scankey routine must consist of three sections as illustrated in the following example**:**

```
USER_SCAN_KEY_INI:
   ; initialization
   ...
   assembly instructions
   ...
   ret
USER_SCAN_KEY:
   ; user scankey main program
   ...
   assembly instructions
   ...
   ; MULT : internal register where trigger number are located
   mov MULT, A
   ret
USER_SCAN_KEY_END:
   ; instructions right before end (sleep mode)
   ret
```

6.8.1.1 "USER_SCAN_KEY_INI:" Label

This label will be called during chip's power-on initialization. Assembly code that requires single initialization may also be written under this label. However, the label should not include any I/O initialization pin parameter.

If no initialization function is required, the label must still be written with "ret" as its single instruction.

6.8.1.2 "USER_SCAN_KEY:" Label

The user main scankey routine is located in this label. This label and its instructions are picked up and processed by EASY compiler. The resulting scan value should be placed in the 'MULT" register and "ret" instruction executed. The resulting value could be either "0" or one of the Trigger Numbers. "0" means trigger is at released position and a Trigger Number means the pertinent trigger is pressed.

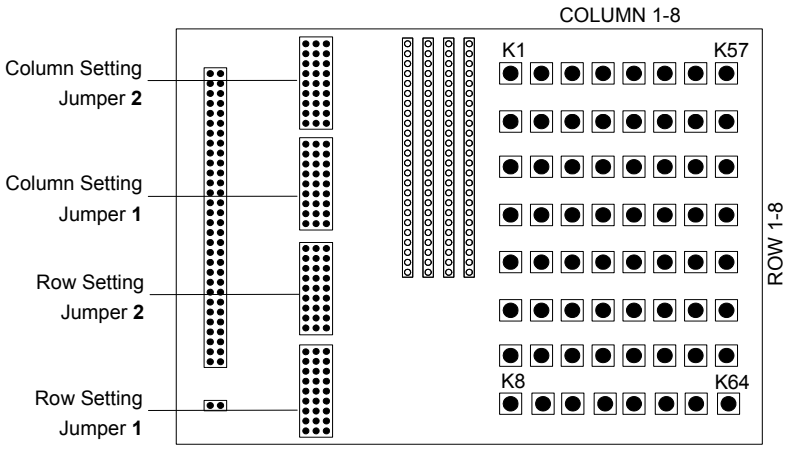
6.8.1.3 "USER_SCAN_KEY_END:" Label

Before EASY lapses into sleep mode, it will call the USER_SCAN_KEY_END label and process any instruction contained therein. If required, you could add your assembly code into this label.

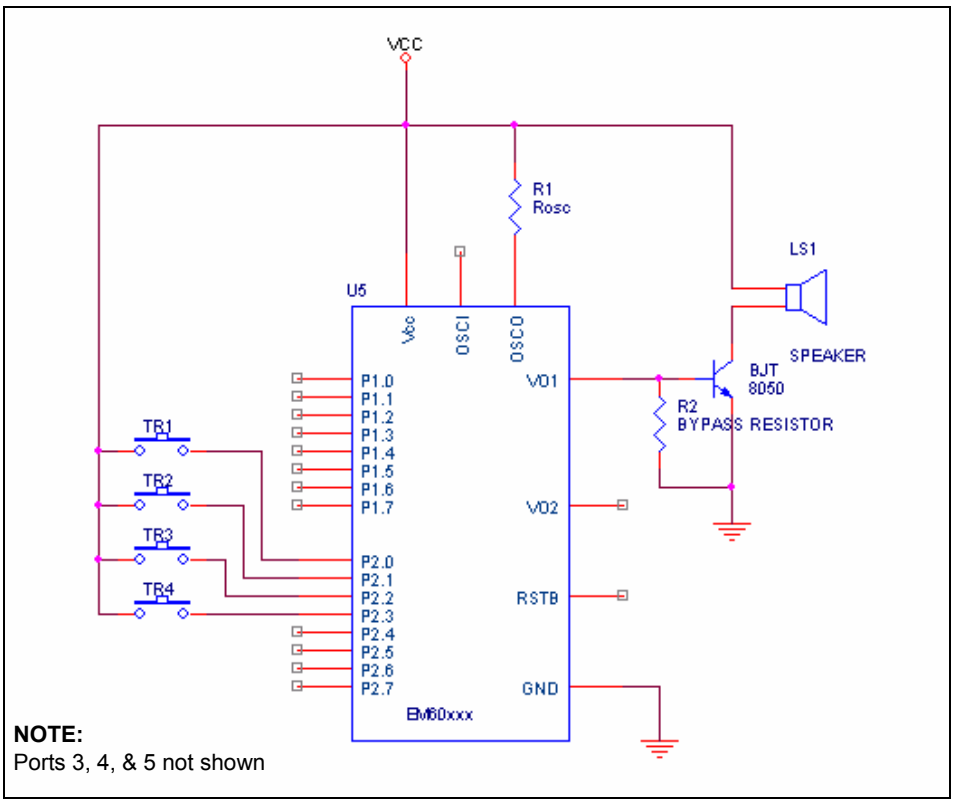
If no END function is required, the label must still be written with "ret" as its single instruction.

6.9 Application Circuits

6.9.1 EMV60KB Jumpers & Keyboard Keys Arrangement



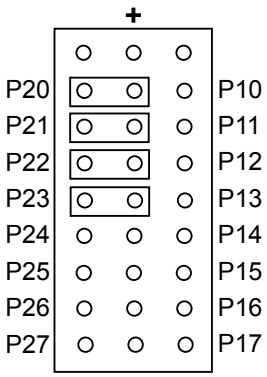
6.9.2 Direct-4 Application Circuit



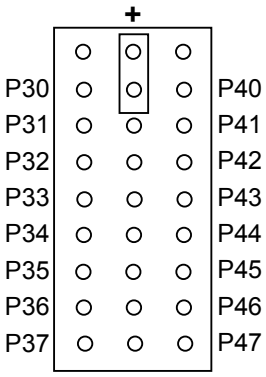
NOTE:
Ports 3, 4, & 5 not shown

6.9.2.1 EMV60KB Jumper Configuration for Direct-4

Use Port 2 as input key. Bridge ROW SETTING JUMPER 1 and COLUMN SETTING JUMPER 2 as follows.



ROW SETTING JUMPER 1

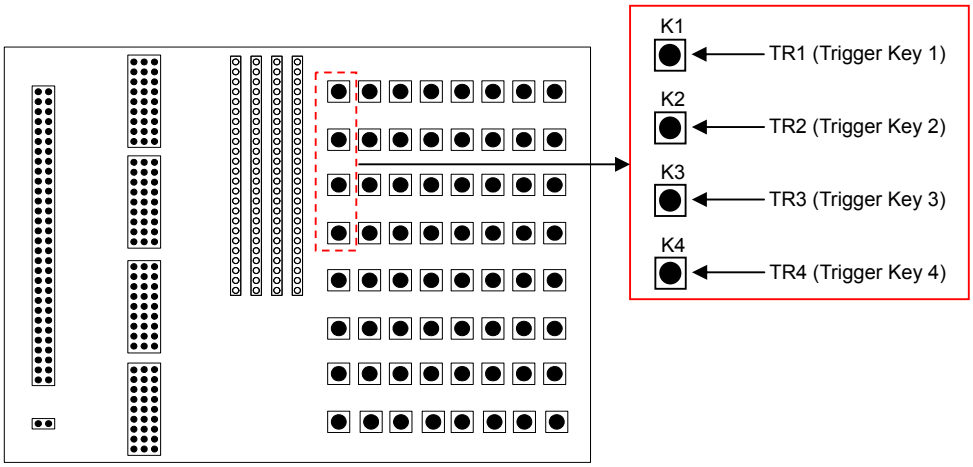


COLUMN SETTING JUMPER 2

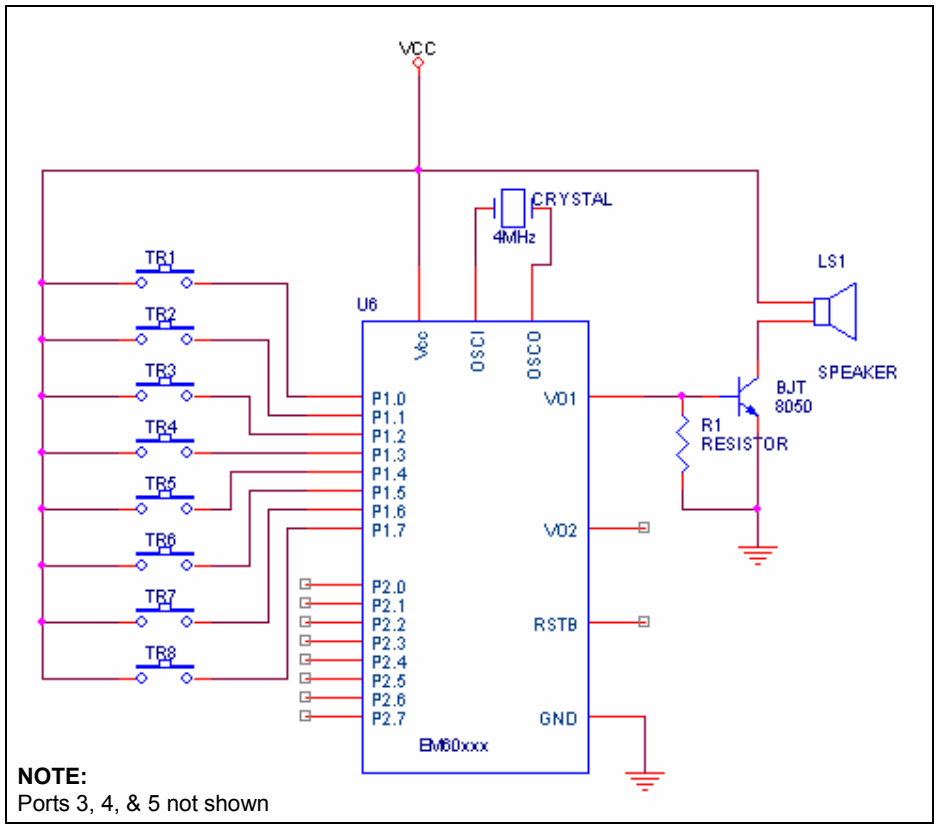
NOTE

*ROW SETTING JUMPER **2** and COLUMN SETTING JUMPER **1** are not used and should be left open.*

6.9.2.2 Applicable EMV60KB Trigger Keys for Direct-4



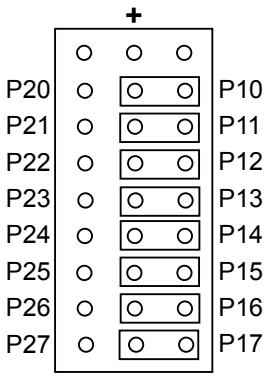
6.9.3 Direct-8 Application Circuit



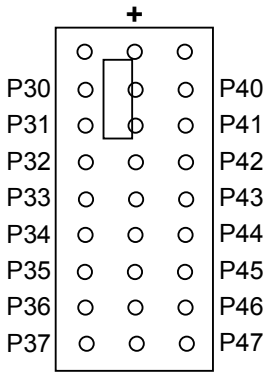
NOTE:
Ports 3, 4, & 5 not shown

6.9.3.1 EMV60KB Jumper Configuration for Direct-8

Use Port 1 as input key. Bridge ROW SETTING JUMPER 1 and COLUMN SETTING JUMPER 2 as follows.



ROW SETTING JUMPER 1

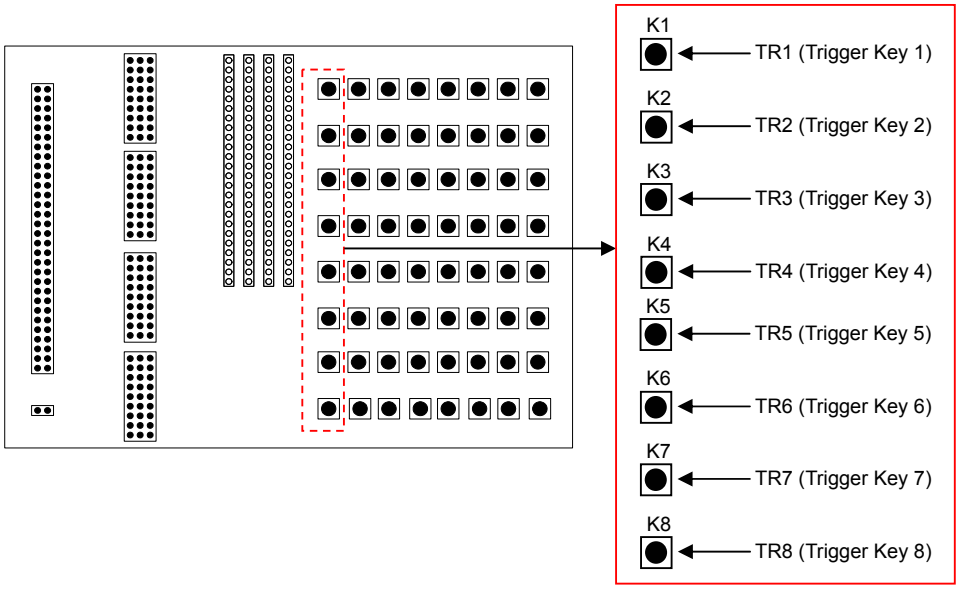


COLUMN SETTING JUMPER 2

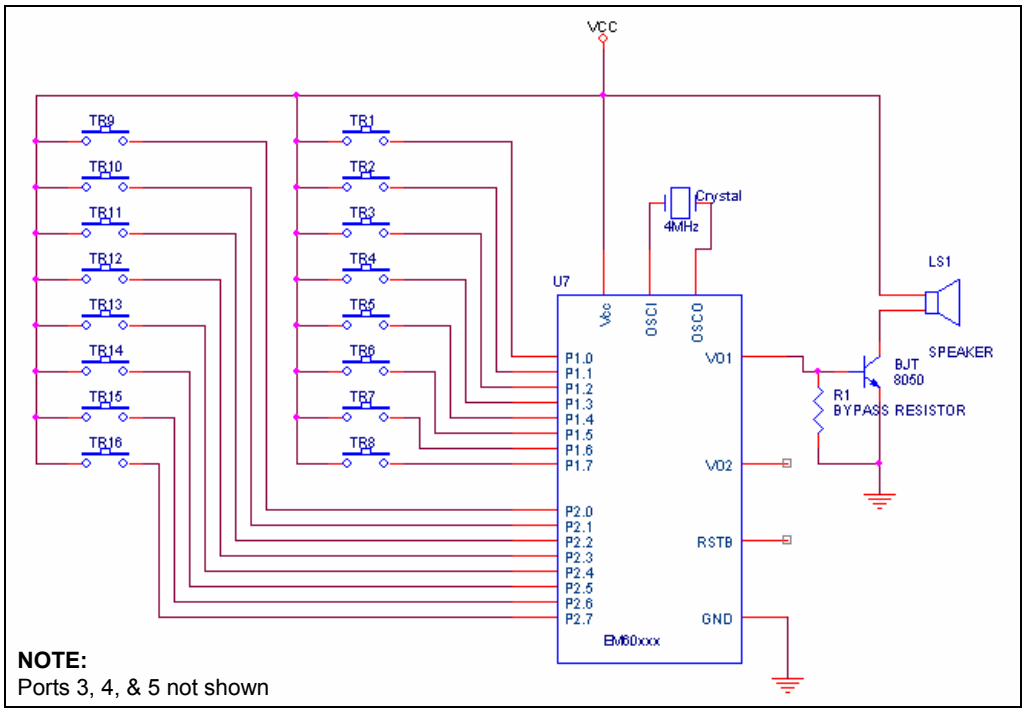
NOTE

*ROW SETTING JUMPER **2** and COLUMN SETTING JUMPER **1** are not used and should be left open.*

6.9.3.2 Applicable EMV60KB Trigger Keys for Direct-8



6.9.4 Direct-9…16 Application Circuit

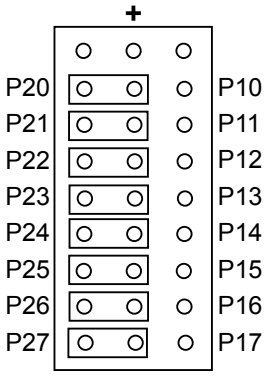


NOTE:
Ports 3, 4, & 5 not shown

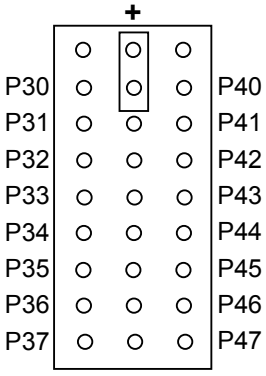
6.9.4.1 EMV60KB Jumper Configuration for Direct-9…16

DIRECT-9…16 use Ports 1 & 2 as input pins for applicable triggers keys which can only be implemented from two EMV60KB keyboards (or from one EMV60KB keyboard and one user provided keyboard). Both sets of keyboards are connected in parallel.

Trigger Keys1…8 use the first EMV60KB with Row & Column jumper settings same with that of DIRECT 8 (see Section 6.9.3.1). Trigger Keys 9…16 use the second EMV60KB and the ROW SETTING JUMPER 1 and COLUMN SETTING JUMPER 2 are bridged as follows.



ROW SETTING JUMPER 1

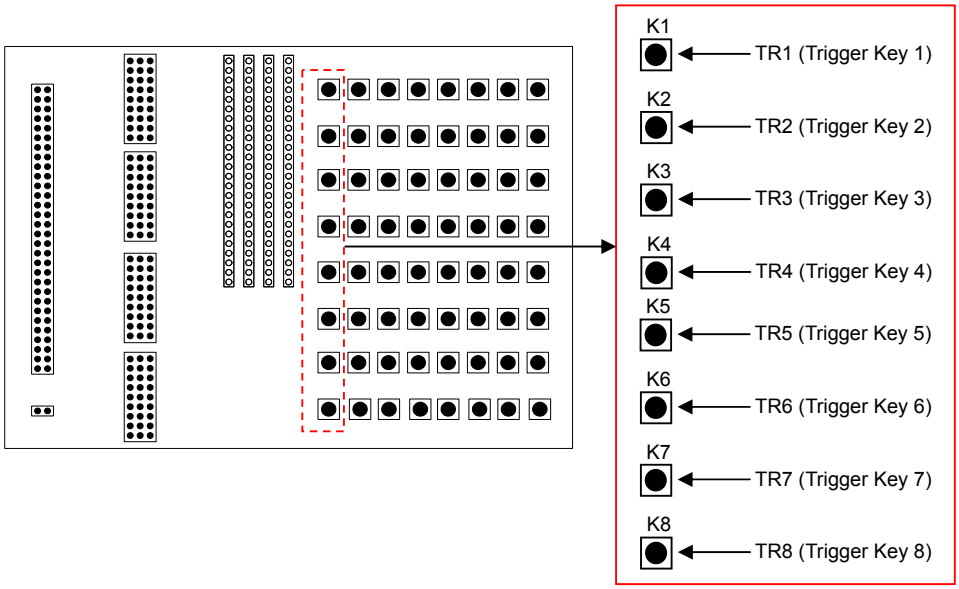


COLUMN SETTING JUMPER 2

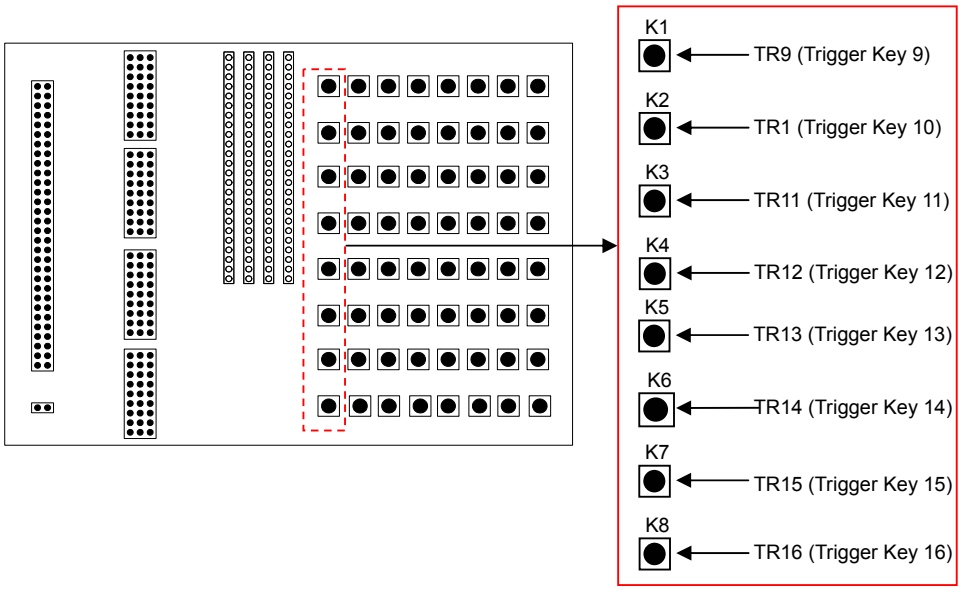
NOTE

*ROW SETTING JUMPER **2** and COLUMN SETTING JUMPER **1** are not used and should be left open.*

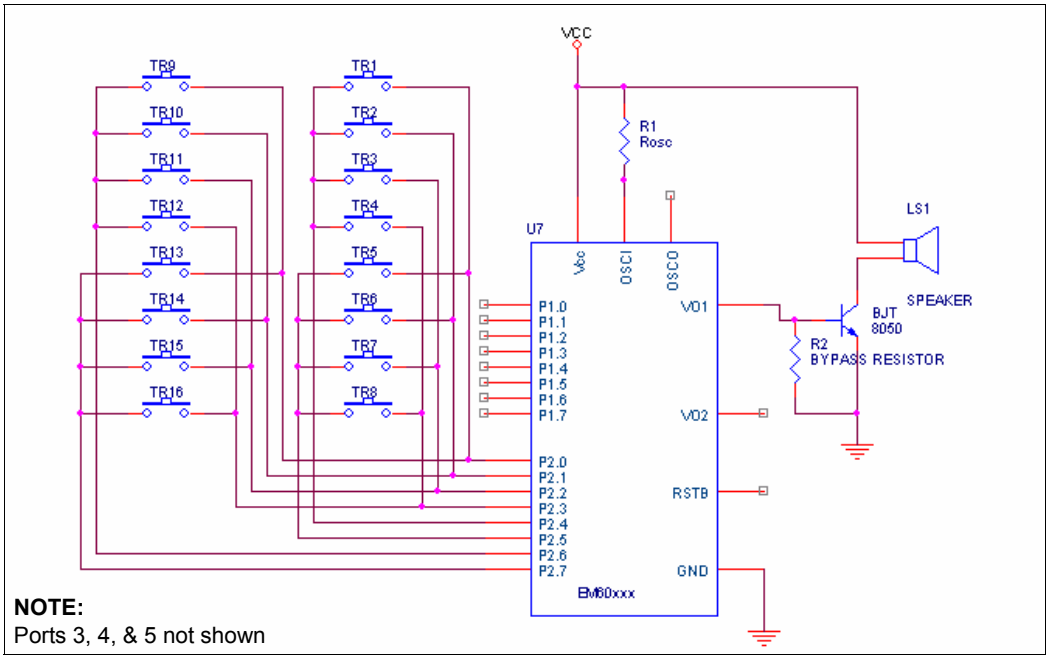
6.9.4.2 First EMV60KB Applicable Trigger Keys 1…8



6.9.4.3 Second EMV60KB Applicable Trigger Keys 9…16



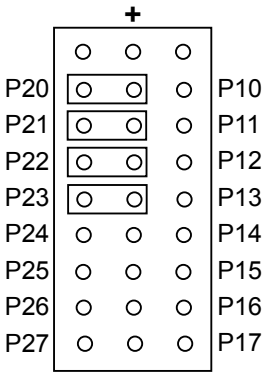
6.9.5 EASY-16 Application Circuit



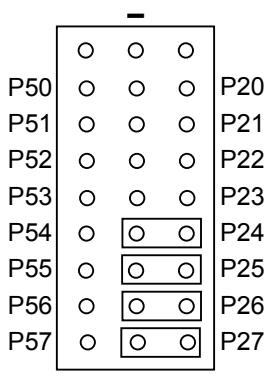
NOTE:
Ports 3, 4, & 5 not shown

6.9.5.1 EMV60KB Jumper Configuration for EASY-16

Use Port 2 as the applicable keyboard scan matrix port. Bridge ROW SETTING JUMPER 1 and COLUMN SETTING JUMPER 1 as follows.



ROW SETTING JUMPER 1

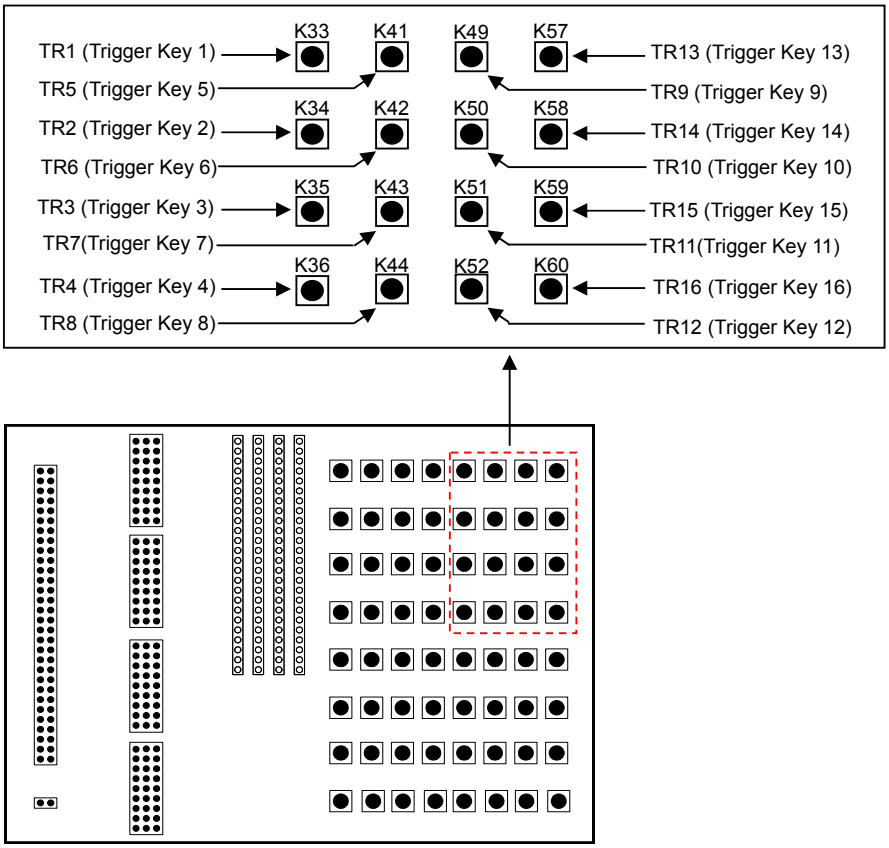


COLUMN SETTING JUMPER 1

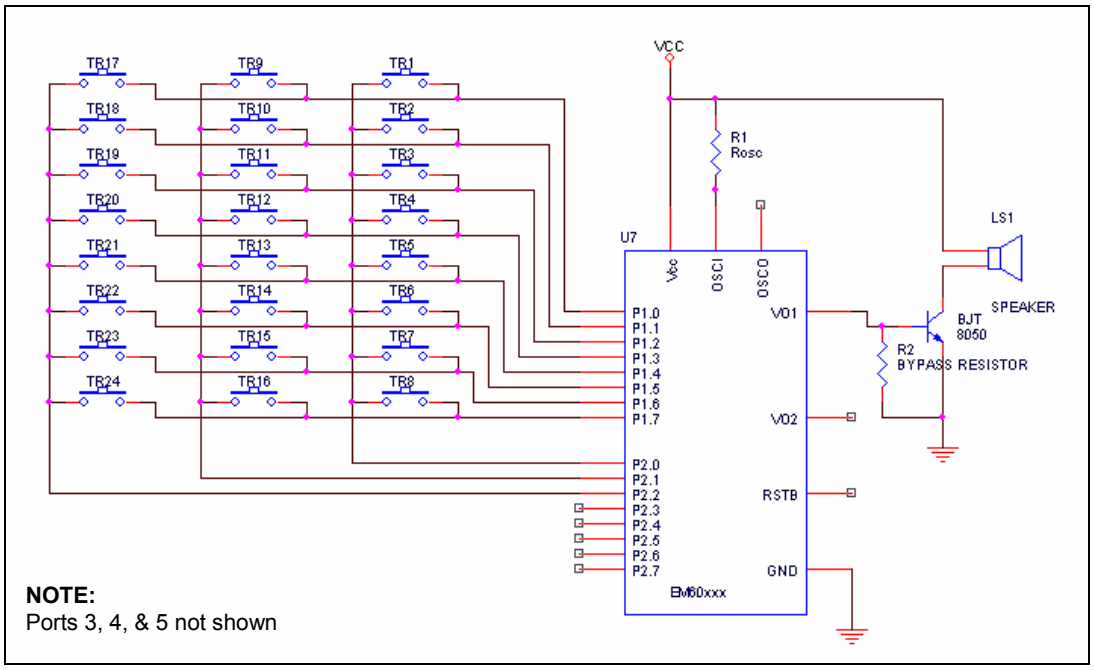
NOTE

*ROW SETTING JUMPER **2** and COLUMN SETTING JUMPER **2** are not used and should be left open.*

6.9.5.2 Applicable EMV60KB Trigger Keys for EASY-16



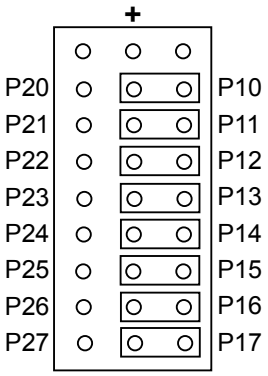
6.9.6 EASY-24 Application Circuit



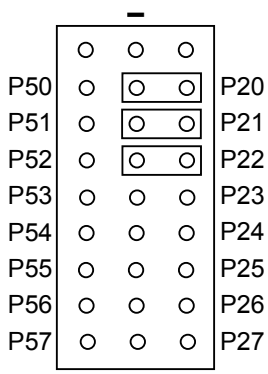
NOTE:
Ports 3, 4, & 5 not shown

6.9.6.1 *EMV60KB Jumper Configuration for EASY-24*

Use Ports 1& 2 as the applicable keyboard scan matrix ports. Bridge ROW SETTING JUMPER 1 and COLUMN SETTING JUMPER 1 as illustrated below.



ROW SETTING JUMPER 1

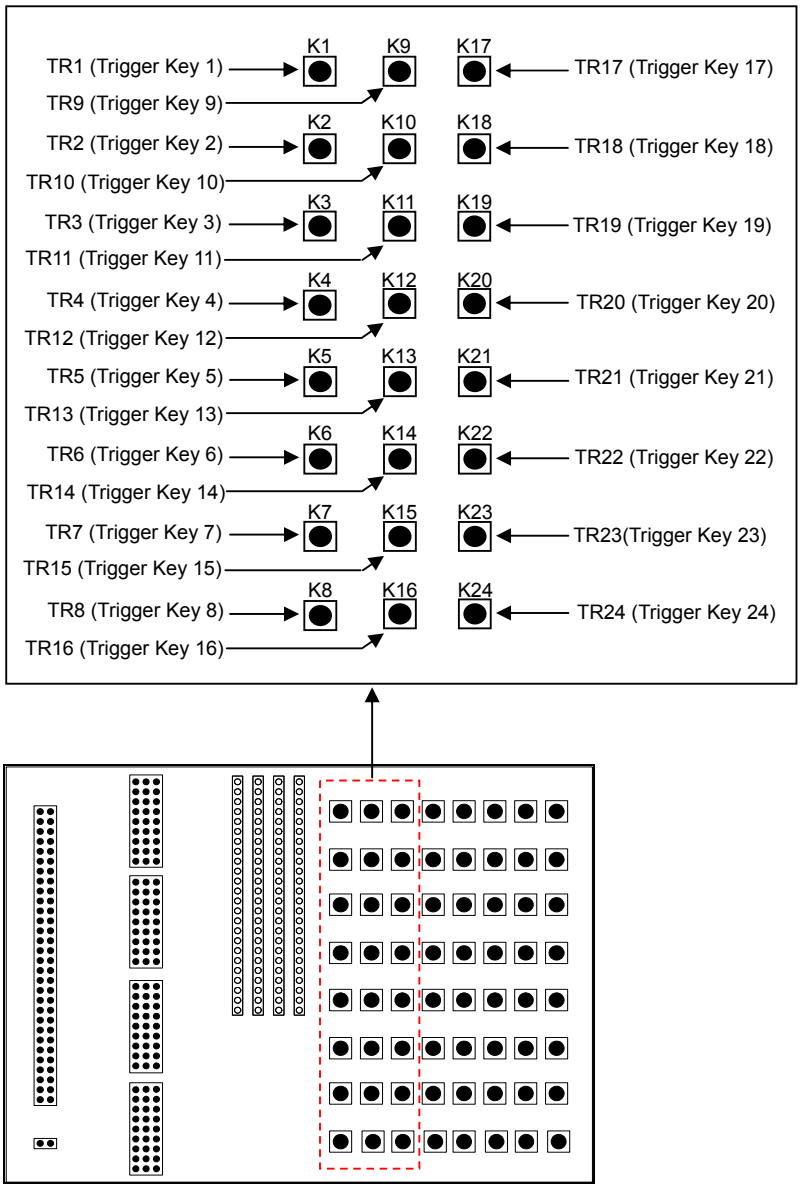


COLUMN SETTING JUMPER 1

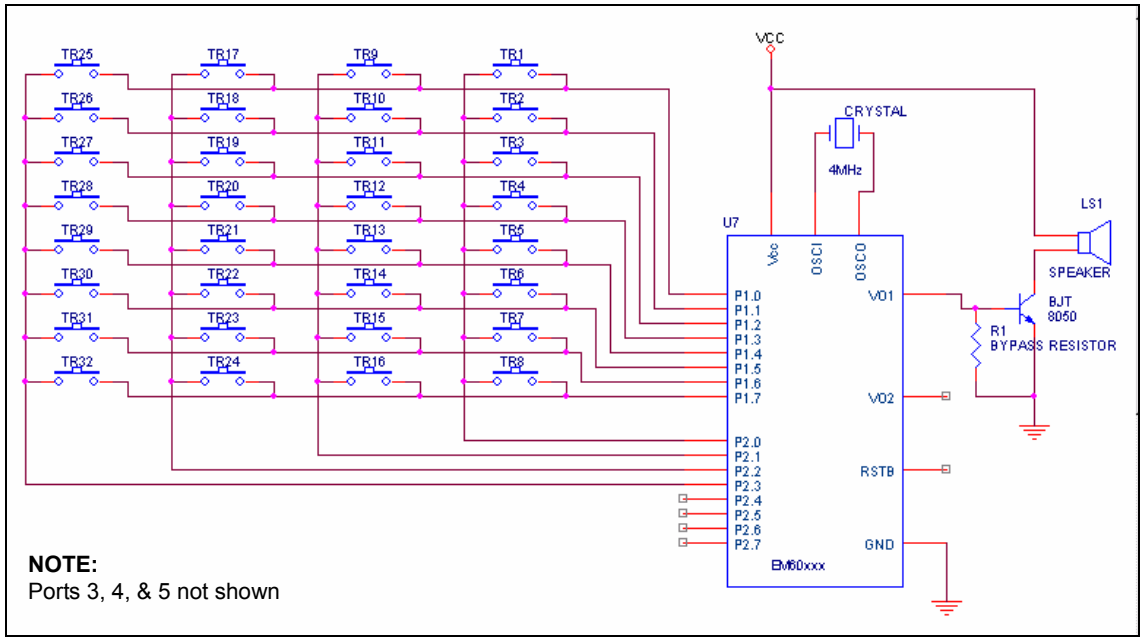
NOTE

*ROW SETTING JUMPER **2** and COLUMN SETTING JUMPER **2** are not used and should be left open.*

6.9.6.2 Applicable EMV60KB Trigger Keys for EAY-24

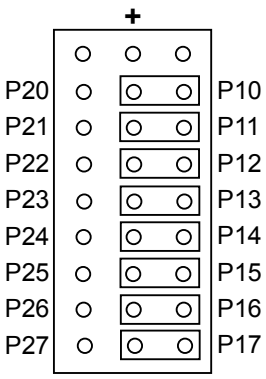


6.9.7 EASY-32 Application Circuit

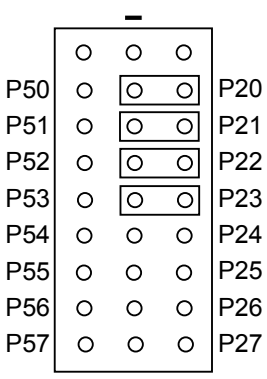


6.9.7.1 *EMV60KB Jumper Configuration for EASY-32*

Use Ports 1& 2 as the applicable keyboard scan matrix ports. Bridge ROW SETTING JUMPER 1 and COLUMN SETTING JUMPER 1 as illustrated below.



ROW SETTING JUMPER 1

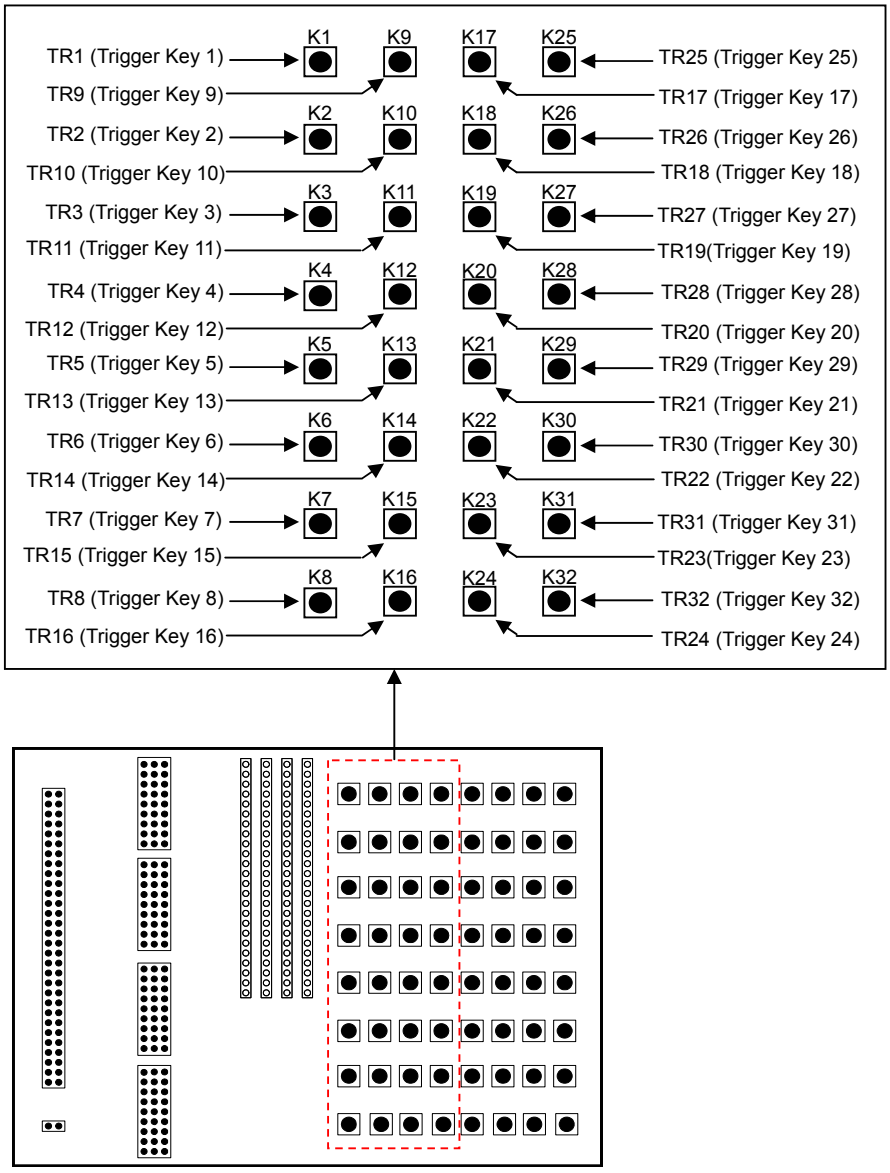


COLUMN SETTING JUMPER 1

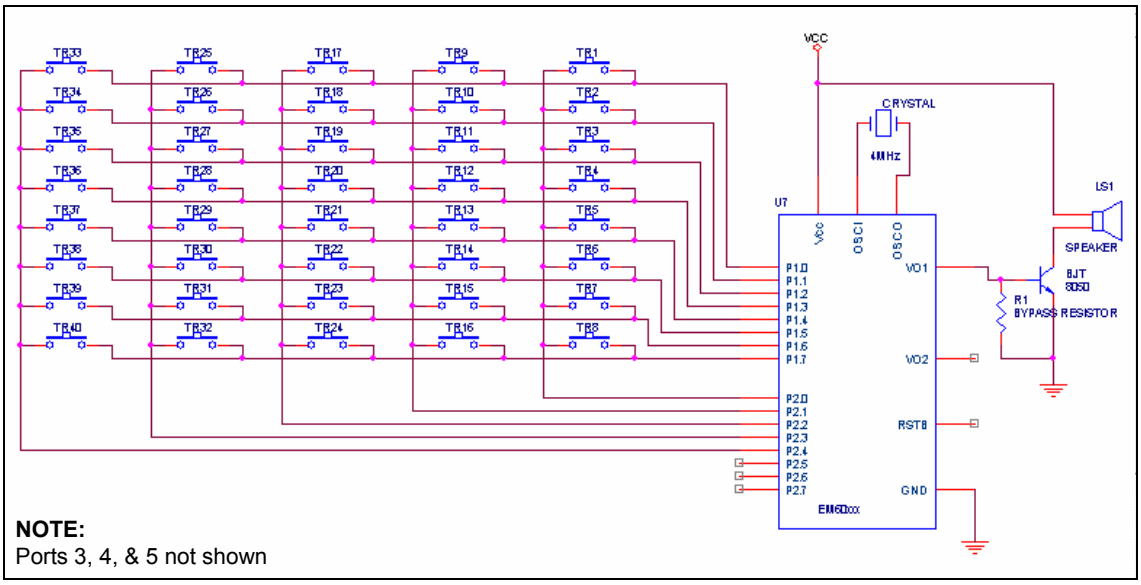
NOTE

*ROW SETTING JUMPER **2** and COLUMN SETTING JUMPER **2** are not used and should be left open.*

6.9.7.2 Applicable EMV60KB Trigger Keys for EAY-32



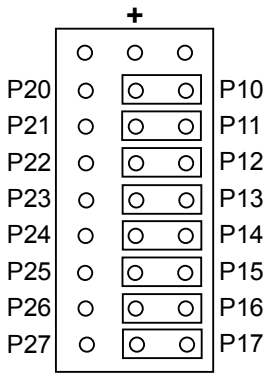
6.9.8 EASY-40 Application Circuit



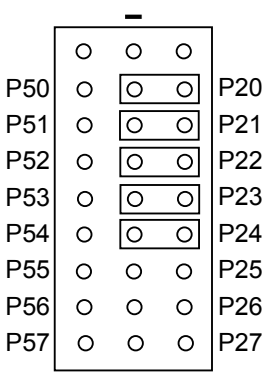
NOTE:
Ports 3, 4, & 5 not shown

6.9.8.1 *EMV60KB Jumper Configuration for EASY-40*

Use Ports 1& 2 as the applicable keyboard scan matrix ports. Bridge ROW SETTING JUMPER 1 and COLUMN SETTING JUMPER 1 as illustrated below.



ROW SETTING JUMPER 1



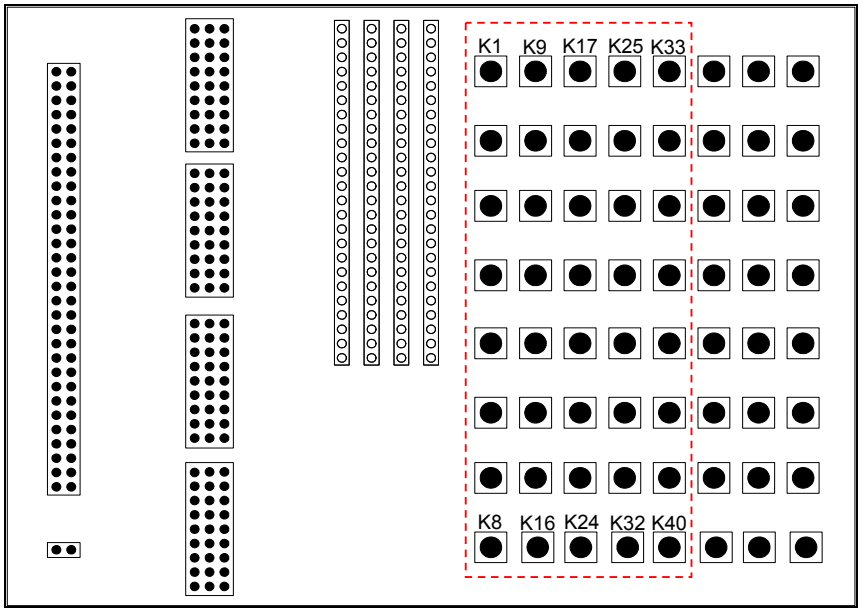
COLUMN SETTING JUMPER 1

NOTE

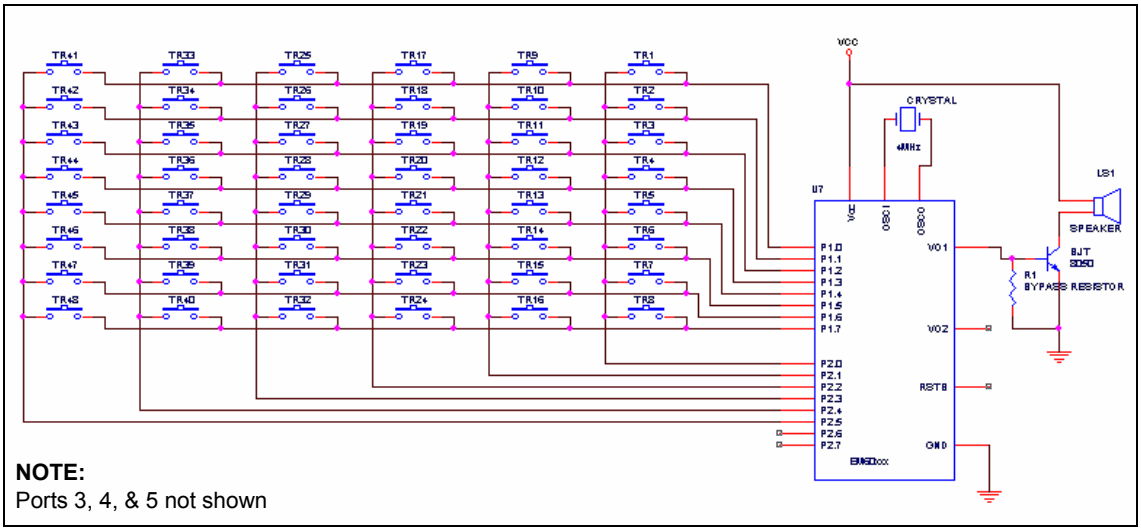
*ROW SETTING JUMPER **2** and COLUMN SETTING JUMPER **2** are not used and should be left open.*

6.9.8.2 Applicable EMV60KB Trigger Keys for EASY-40

Applicable keyboard keys are similar with EASY-32 (Section 6.9.7.2) except applicable keys are extended to include Keys K33 to K40 as indicated below.



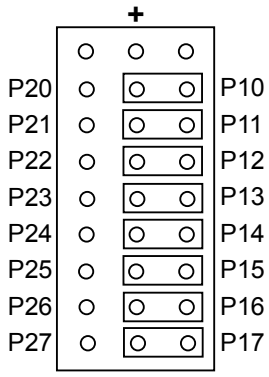
6.9.9 EASY-48 Application Circuit



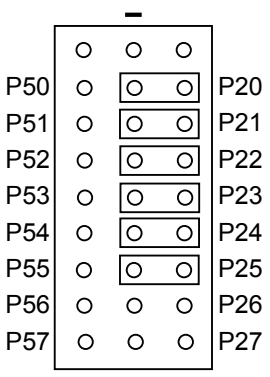
NOTE:
Ports 3, 4, & 5 not shown

6.9.9.1 EMV60KB Jumper Configuration for EASY-48

Use Ports 1& 2 as the applicable keyboard scan matrix ports. Bridge ROW SETTING JUMPER 1 and COLUMN SETTING JUMPER 1 as illustrated below.



ROW SETTING JUMPER 1



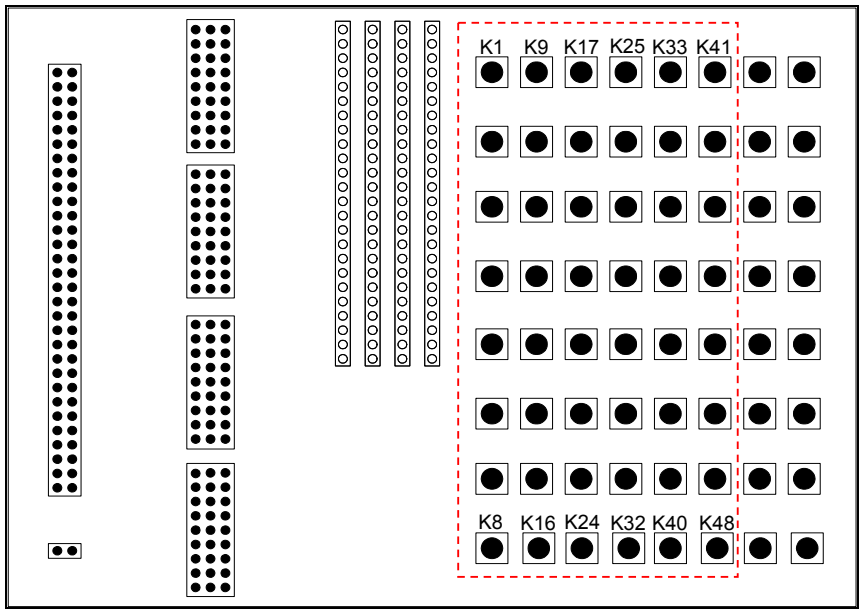
COLUMN SETTING JUMPER 1

NOTE

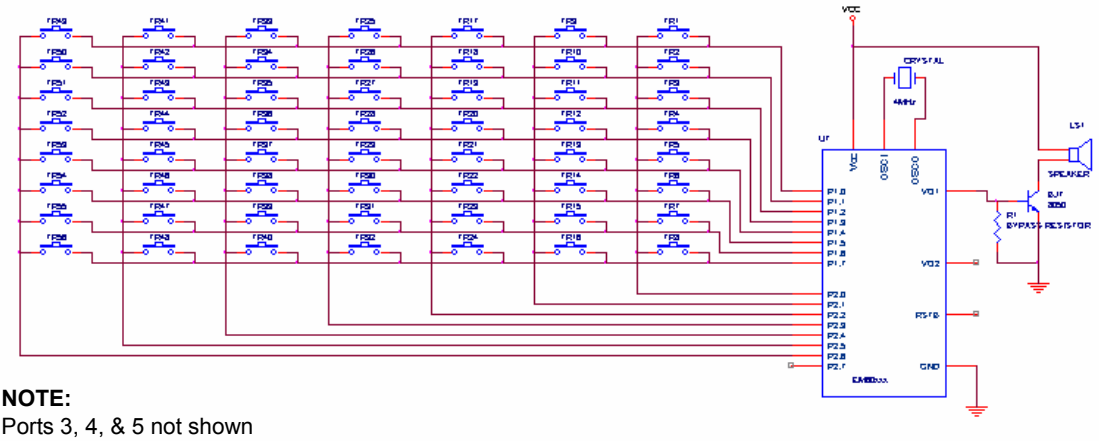
*ROW SETTING JUMPER **2** and COLUMN SETTING JUMPER **2** are not used and should be left open.*

6.9.9.2 Applicable EMV60KB Trigger Keys for EASY-48

Applicable keyboard keys are similar with EASY-32 (Section 6.9.7.2) except applicable keys are extended to include Keys K33 to K48 as indicated below.



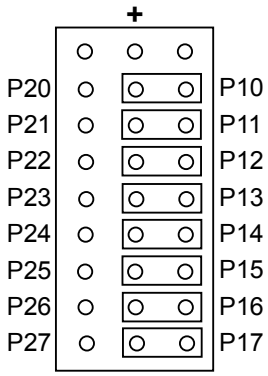
6.9.10 EASY-56 Application Circuit



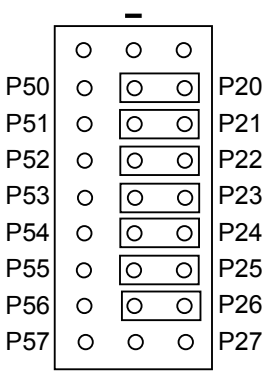
NOTE:
Ports 3, 4, & 5 not shown

6.9.10.1 EMV60KB Jumper Configuration for EASY-56

Use Ports 1& 2 as the applicable keyboard scan matrix ports. Bridge ROW SETTING JUMPER 1 and COLUMN SETTING JUMPER 1 as illustrated below.



ROW SETTING JUMPER 1



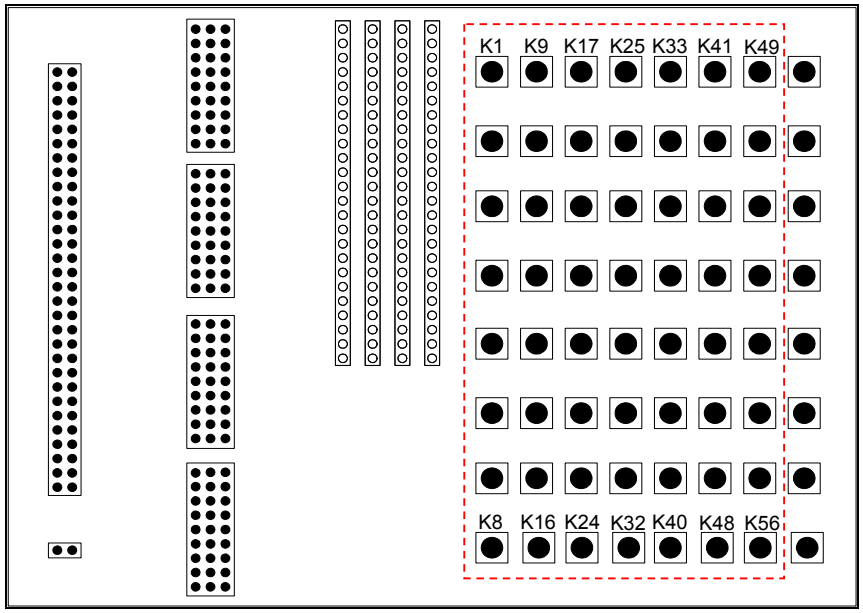
COLUMN SETTING JUMPER 1

NOTE

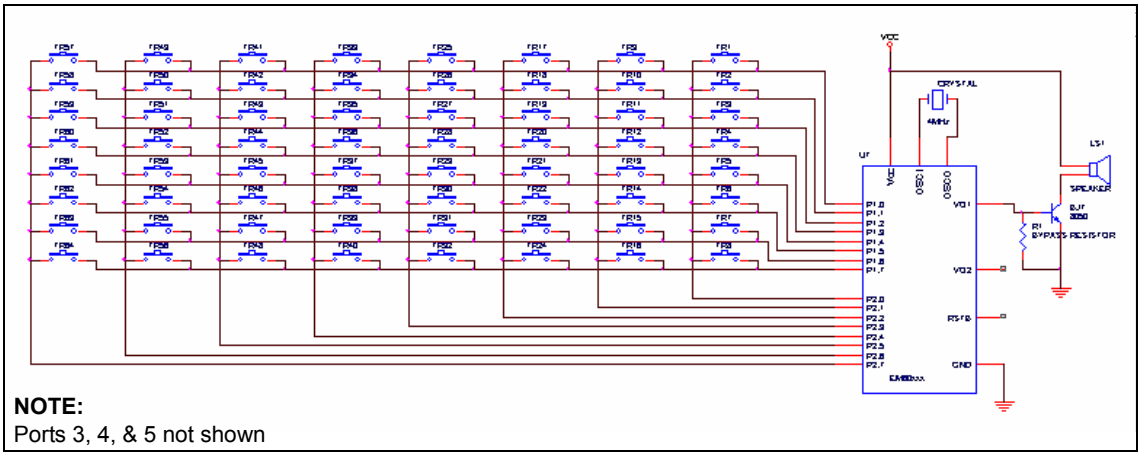
*ROW SETTING JUMPER **2** and COLUMN SETTING JUMPER **2** are not used and should be left open.*

6.9.10.2 Applicable EMV60KB Trigger Keys for EASY-56

Applicable keyboard keys are similar with EASY-32 (Section 6.9.7.2) except applicable keys are extended to include Keys K33 to K56 as indicated below.



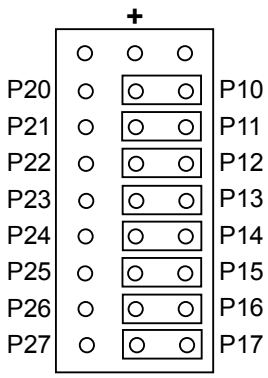
6.9.11 EASY-64 Application Circuit



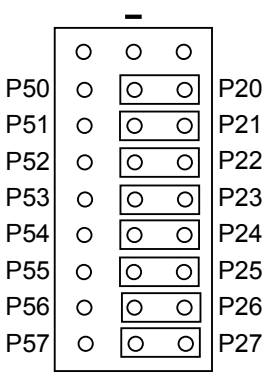
NOTE:
Ports 3, 4, & 5 not shown

6.9.11.1 EMV60KB Jumper Configuration for EASY-64

Use Ports 1& 2 as the applicable keyboard scan matrix ports. Bridge ROW SETTING JUMPER 1 and COLUMN SETTING JUMPER 1 as illustrated below.



ROW SETTING JUMPER 1



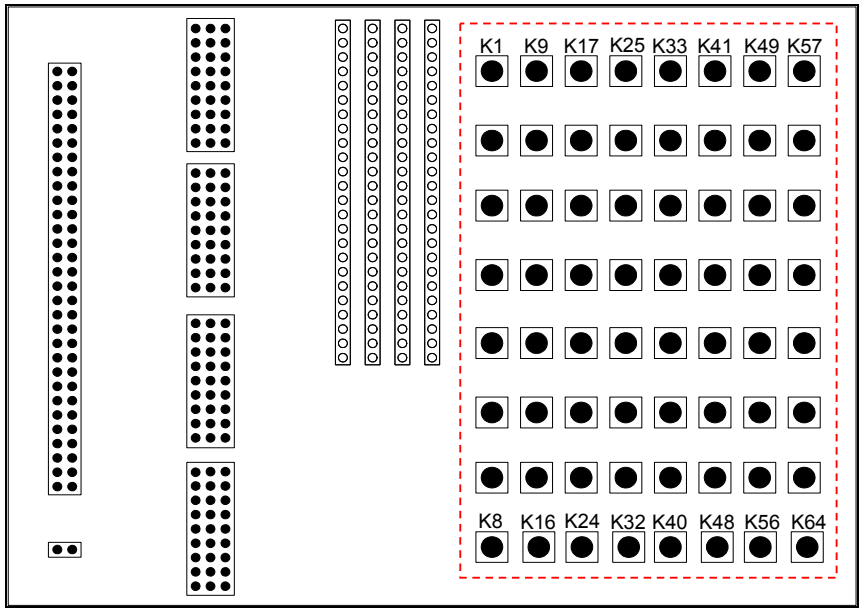
COLUMN SETTING JUMPER 1

NOTE

*ROW SETTING JUMPER **2** and COLUMN SETTING JUMPER **2** are not used and should be left open.*

6.9.11.2 Applicable EMV60KB Trigger Keys for EASY-64

Applicable keyboard keys are similar with EASY-32 (Section 6.9.7.2) except applicable keys are extended to include all Keys K1 to K64 as indicated below.



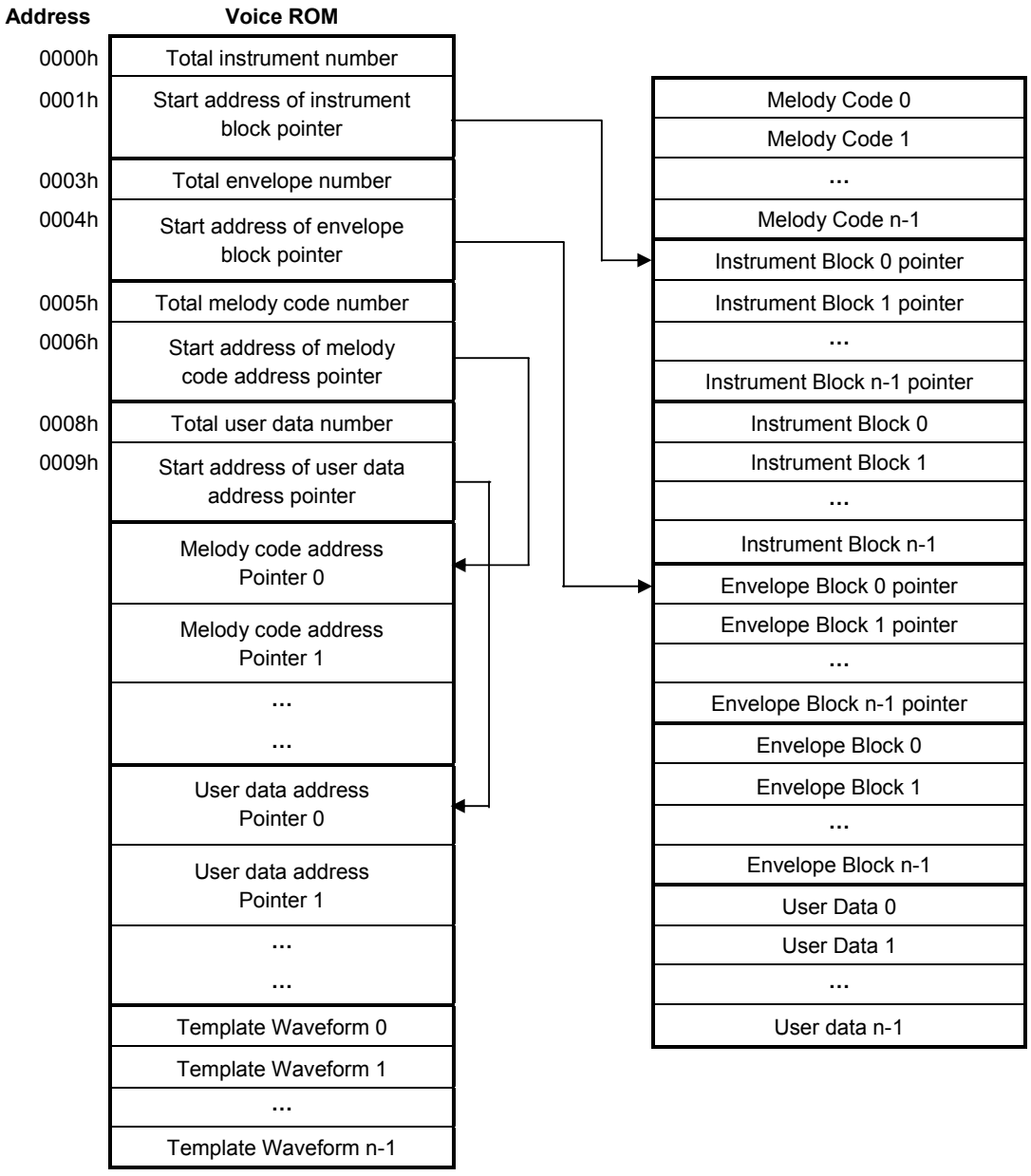
Appendix A
Voice ROM Data Allocation

A.1 Introduction

When compiling, the Voice Builder collects the speech/melody/data codes and writes them to a Voice ROM data file. The information in Voice ROM contains Instrument Block, Instrument Block Pointer, Envelope Block, Envelope Block Pointer, Melody Code, Melody Code Address Pointer, User Data, and User Address Pointer. Each starting address of pointer and each block number are located at the beginning of Voice ROM, followed by Instrument, Envelope, Melody, and User Data.

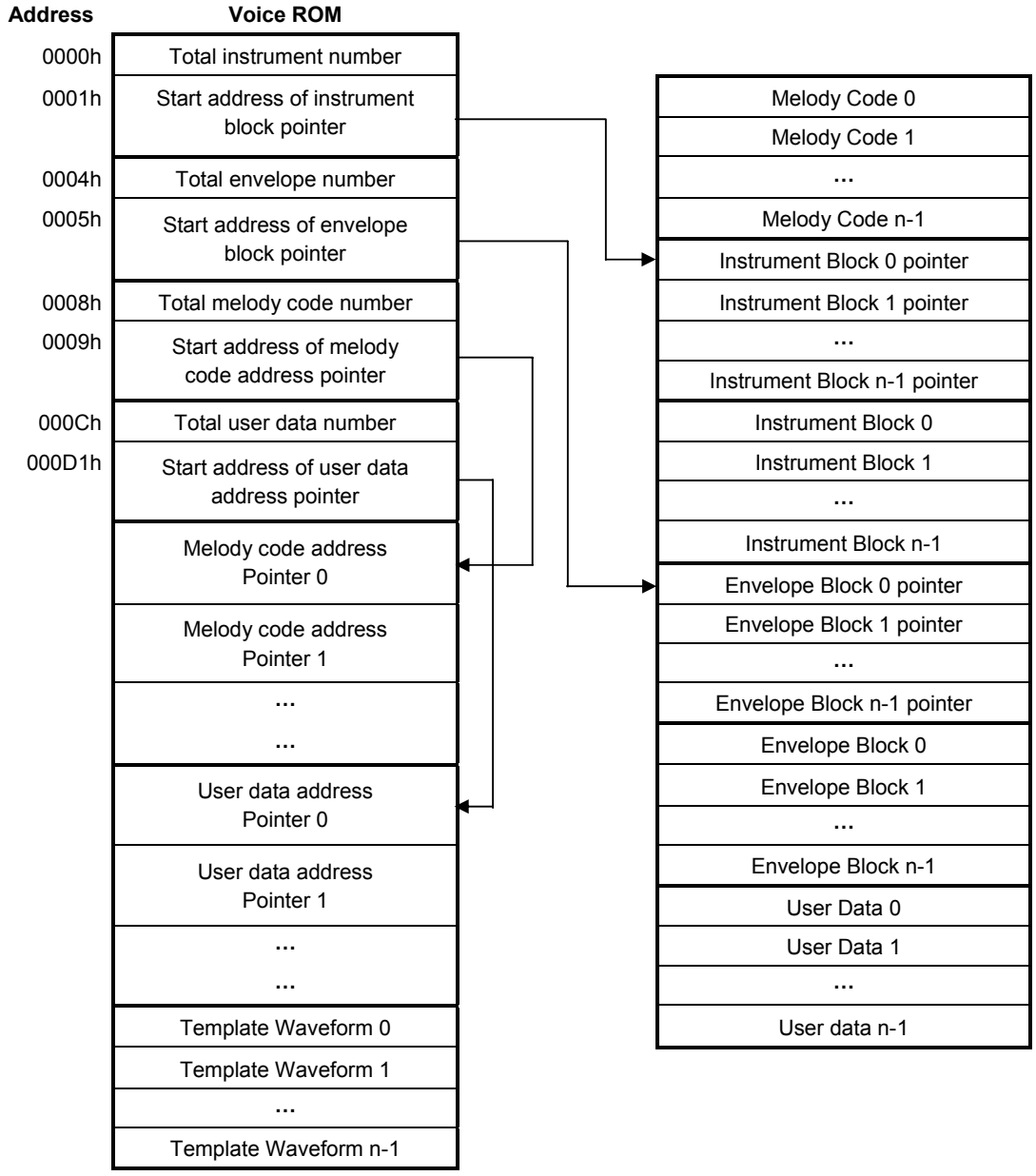
A.2 EM61001/002/100 Voice ROM Configuration

The length for each pointer is 2 bytes long.



A.3 EM61200/300/400/500 Voice ROM Configuration

The length for each pointer is 3 bytes long.



A.4 Data Table Address for Include File

The starting address of each pointer is defined in the EM61IDE Program generated file **.adr* . This file contains the Total Instrument Number, Instrument Block Pointer, Total Envelope Number, Melody Code Pointer, Total User Data, and the User Data Pointer. These are illustrated in the example below**:**

A.4.1 For EM61001/002/100

The length of each pointer is 2 bytes. *Midi_file0 ... userdata_file0 ..* is the filename of the MIDI file or of the user data you have added.

```
inst_number == n
inst_block == 0x1234
inst_block_L == 0x34
inst_block_H == 0x12
env_number == m
env_block == 0x5678
env_block_L == 0x78
env_block_H == 0x56
melody_number == k
melody_code == 0xABCD
melody_code_L == 0xCD
melody_code_H == 0xAB
midi_file0 == 0xBBCD
midi_file0_L == 0xCD
midi_file0_H == 0xBB
…
userdata_number == l
userdata == 0xCF12
userdata_L == 0x12
userdata_H == 0xCF
userdata_file0 == 0xDF34
userdata_file0_L == 0x34
userdata_file0_H == 0xDF
…
```

A.4.2 For EM61200/300/400/500

The length of each pointer is 2 bytes. *Midi_file0 ... userdata_file0 ..* is the filename of MIDI file or of the user data you have added.

```
inst_number == n
inst_block == 0x123456
inst_block_L == 0x56
inst_block_M == 0x34
inst_block_H == 0x12
env_number == m
env_block == 0x56789A
env_block_L == 0x9A
env_block_M == 0x78
env_block_H == 0x56
melody_number == k
melody_code == 0xABCD12
melody_code_L == 0x12
melody_code_M == 0xCD
melody_code_H == 0xAB
midi_file0 == 0xBBCD34
midi_file0_L == 0x34
midi_file0_M == 0xCD
midi_file0_H == 0xBB
…
userdata_number == l
userdata == 0xEF1234
userdata_L == 0x34
userdata_M == 0x12
userdata_H == 0xEF
userdata_file0 == 0xEF1234
userdata_file0_L == 0x34
userdata_file0_M == 0x12
userdata_file0_H == 0xEF
```

Appendix B
MIDI (EM61.mid) File

B.1 Compatible MIDI File Format

MIDI files must be compatible with the General Midi (GM) specification and saved as “MIDI Format 1” type files.

Saving MIDI as “MIDI Format 1”

NOTE

The above figure as well all other figures used in this Appendix are from “CakeWalk” MIDI editor.

B.2 MIDI Duration

B.2.1 Acceptable Minimum Beat

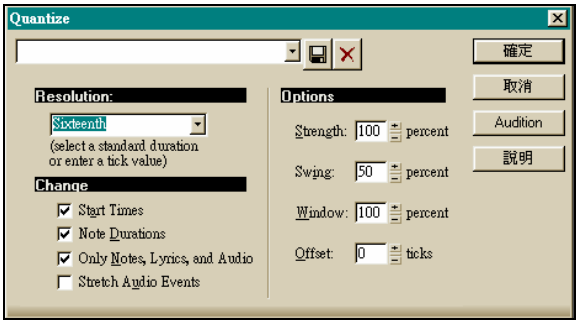
1/16 beat (64^{th} – note; 64^{th}- rest)

1/12 beat (32^{nd} Triplet – note; 32^{nd} Triplet – rest)

Acceptable length should be a multiple of the above beats.

B.2.2 Quantization and Resolution

Use the quantization feature to adjust notes position and apply the shortest beat (see Section C.2.1 above) as the resolution value.



Using “CakeWalk” Quantize Dialog for Quantization

B.2.3 Timebase

It is recommended that the Timebase of your editing software is set at *1 beat = 48 ticks* in order to facilitate debugging.

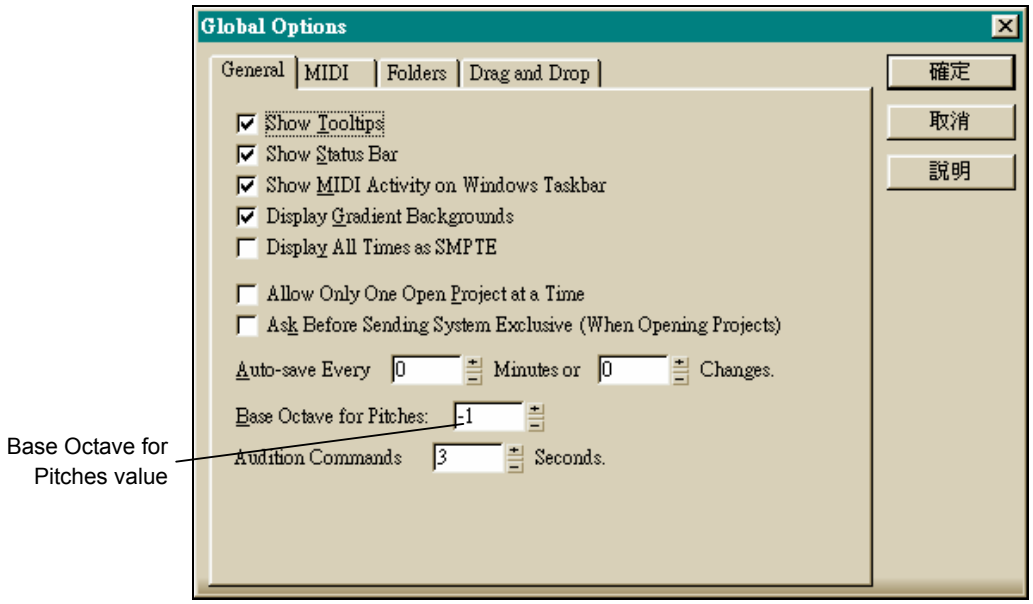
B.3 Frequency

B.3.1 Note

Each Note in the MIDI is numbered from 0 to 127 with C = 60 as the Mid-Note. The ability of the current version of EM61000 in recognizing pitch is unstable. Refer to *61_tonelib.doc* for further details.

B.3.2 Base Octave

It is recommended that the setting for Base Octave for pitches is set at C4 = 60 to facilitate debugging (under Cakewalk, when Base Octave for Pitches is –1, C4 = 60; when Base Octave for Pitches is 0, C5 = 60)



Base Octave for Pitches Setting with “CakeWalk” Global Options Dialog

B.4 Patch and Channel in EM61000

B.4.1 EM61000 Capabilities

The General MIDI (GM) is specified to directly access up to 128 patches of musical instruments (numbered 0 ~ 127), 47 patches of drum and percussion parts (numbered 35 ~ 81), and 16 channels (numbered 1 ~ 16). The current version of EM61000 IDE software supports the GM requirements except for the patches of musical instrument. Presently, the EM61000 Tone Library (*INSLIB*) only supports 66 patches. User needs to modify the MIDI file in order to fill the missing instrument patches to make it compatible with EM61000.

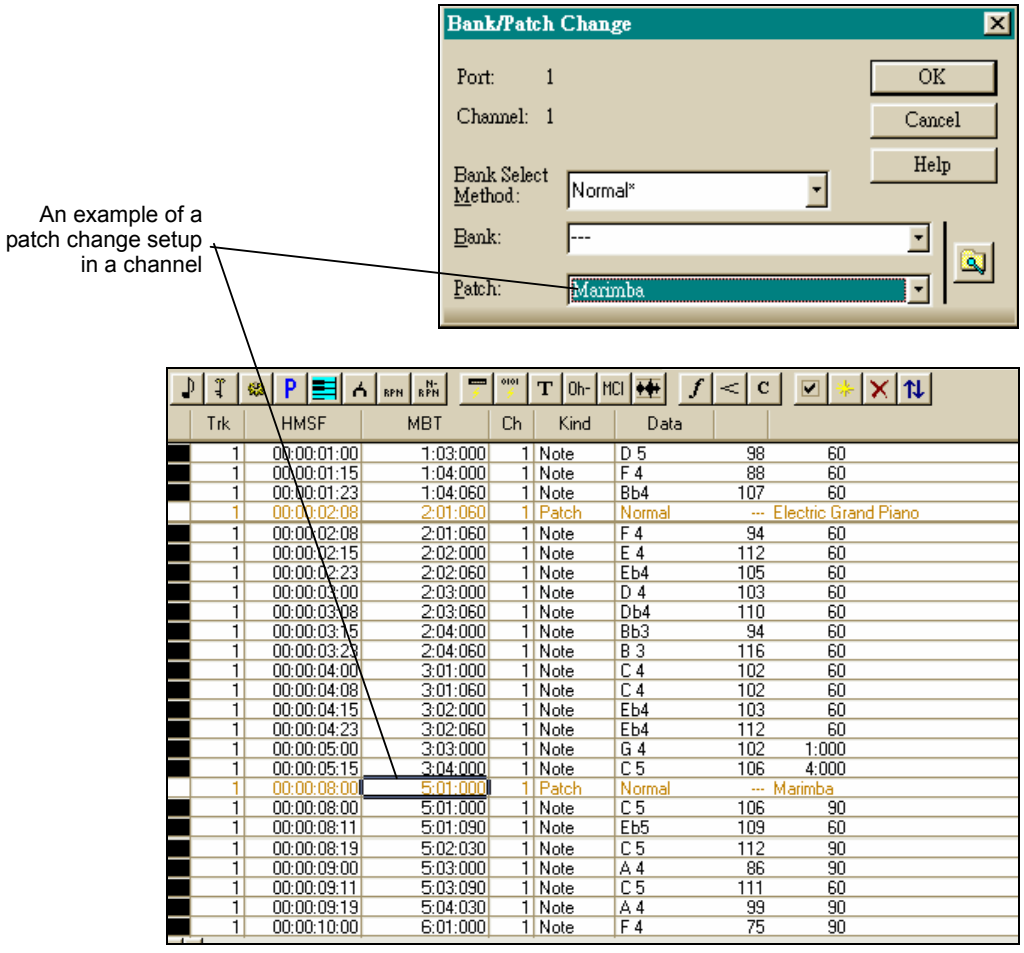
B.4.2 Maximum Tones in Channel and Looping

The maximum number of tones that can be played at a given moment with EM61000 channel is 8.

EM61000 can only accept 4 channels of long loop tones when each loop exceeds 128 bytes.

B.4.3 Patch Change

EM61000 supports multiple patch changes in a channel.



Patch Change Setup in a Channel under “CakeWalk”

B.4.3 General MIDI vs. EM61000 Patches

The following table shows patches where EM61000 complies with the 128 preset MIDI specified instruments. Note that Patches Number 115, 116, and 117 are unique to EM61000 and do not conform to MIDI specified instruments.

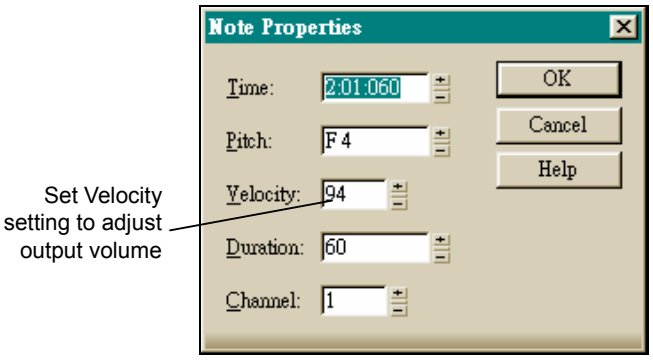
Patch No.	General MIDI Instrument Name	EM61000	Patch No.	General MIDI Instrument Name	EM61000	Patch No.	General MIDI Instrument Name	EM61000
000	Acoustic Grand Piano	Comply	043	Contrabass	Comply	086	Lead 7 (Brit Saw Wave	–
001	Bright Acoustic Piano	Comply	044	Tremolo Strings	Comply	087	Lead 8 (Brass + Lead)	–
002	Electric Grand Piano	–	045	Pizzicato Strings	Comply	088	Pad 1 (Fantasia)	Comply
003	Honky-Tonk Piano	Comply	046	Orchestral Harp	Comply	089	Pad 2 (Warm)	–
004	Rhodes Piano	Comply	047	Timpani	–	090	Pad 3 (Polysynth)	–
005	Chorused Piano	Comply	048	String Ensemble 1	–	091	Pad 4 (Choir)	Comply
006	Harpsichord	Comply	049	String Ensemble 2	–	092	Pad 5 (Bowed Glass)	–
007	Clavinet	Comply	050	Synth Strings 1	–	093	Pad 6 (Metallic)	–
008	Celesta	Comply	051	Synth Strings 2	–	094	Pad 7 (Halo)	–
009	Glockenspiel	–	052	Choir Aahs	–	095	Pad 8 (Sweep)	–
010	Music Box	Comply	053	Voice Oohs	–	096	SFX 1 (Ice Rain)	–
011	Vibraphone	Comply	054	Synth Voice	–	097	SFX 2 (Sound Track)	–
012	Marimba	Comply	055	Orchestra Hit	–	098	SFX 3 (Crystal)	–
013	Xylophone	Comply	056	Trumpet	Comply	099	SFX 4 (Atmosphere)	–
014	Tubular Bells	Comply	057	Trombone	Comply	100	SFX 5 (Brightness)	–
015	Dulcimer	Comply	058	Tuba	Comply	101	SFX 6 (Goblins)	–
016	Hammond Organ	–	059	Muted Trumpet	Comply	102	SFX 7 (Echo Drops)	–
017	Percussive Organ	Comply	060	French Horn	Comply	103	SFX 8 (Sci- Fiction)	–
018	Rock Organ	Comply	061	Brass Section	–	104	Sitar	–
019	Church Organ	Comply	062	Synth Brass 1	–	105	Banjo	Comply
020	Reed Organ	–	063	Synth Brass 2	–	106	Shamisen	–
021	Accordion	Comply	064	Soprano Sax	Comply	107	Koto	–
022	Harmonica	Comply	065	Alto Sax	–	108	Kalimba	–
023	Tango Accordion	Comply	066	Tenor Sax	Comply	109	Bagpipe	–
024	Acoustic Nylon Guitar	Comply	067	Baritone Sax	Comply	110	Fiddle	–
025	Acoustic Steel Guitar	Comply	068	Oboe	Comply	111	Shanai	–
026	Electric Jazz Guitar	Comply	069	English Horn	Comply	112	Timkie Bell	–
027	Electric Clean Guitar	Comply	070	Bassoon	Comply	113	Agogo	–
028	Electric Muted Guitar	Comply	071	Clarinet	Comply	114	Steel Drum	Comply
029	Overdriven Guitar	Comply	072	Piccolo	Comply	115	Woodblock	* Major Triad
030	Distortion Guitar	Comply	073	Flute	Comply	116	Taiko Drum	* Minor Triad
031	Guitar Harmonics	Comply	074	Recorder	Comply	117	Melodic Tone	* Mandolin
032	Acoustic Bass	Comply	075	Pan Flute	Comply	118	Synth Drum	–
033	Electric Bass Fingered	Comply	076	Bottle Blow	Comply	119	Reverse Cymbal	–
034	Electric Bass Picked	Comply	077	Shakuhachi	Comply	120	Guitar Fret Noise	–
035	Fretless Bass	–	078	Whistle	–	121	Breath Noise	–
036	Slap Bass1	Comply	079	Ocarina	–	122	Seashore	–
037	Slap Bass2	Comply	080	Lead 1 (Square Wave	–	123	Bird Tweet	–
038	Synth Bass1	Comply	081	Lead 2 (Saw Tooth)	–	124	Telephone Ring	–
039	Synth Bass2	Comply	082	Lead 3 (Caliope)	–	125	Helicopter	–
040	Violin	Comply	083	Lead 4 (Chiff)	–	126	Applause	–
041	Viola	Comply	084	Lead 5 (Charang)	–	127	Gunshot	–
042	Cello	Comply	085	Lead 6 (Solo Voice)	–			

** 115, 116 & 117 are EM61000 unique instruments*

B.5 Volume

B.5.1 EM61000 Playing Volume

Modifying the velocity in the MIDI file will change the EM61000 overall playing volume.



Modifying Play Volume under "CakeWalk"

B.5.2 Volume Range

The acceptable volume range for EM61000 is 0 ~ 127.

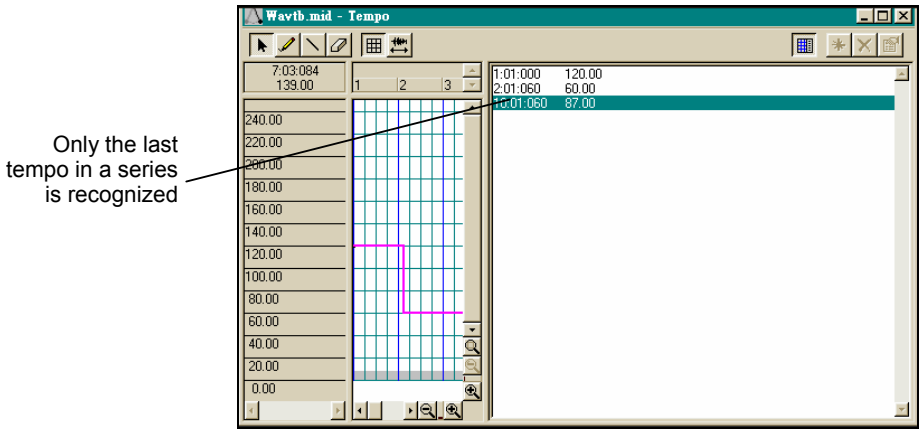
B.6 Tempo

B.6.1 Tempo Range

The acceptable tempo range for EM61000 is 40 ~ 240.

B.6.2 Tempo Limitation

EM61000 Voice Translator can only accept one tempo for each MIDI. If several tempos are available, only the last tempo is accepted.



Tempo Range Setup

B.6.3 Multiple Tempos

EM61000 can only accept one tempo in a MIDI. If more than one tempo is required, chunk the tempo into several parts and place each part into a separate MIDI file. Then program them to play continuously.

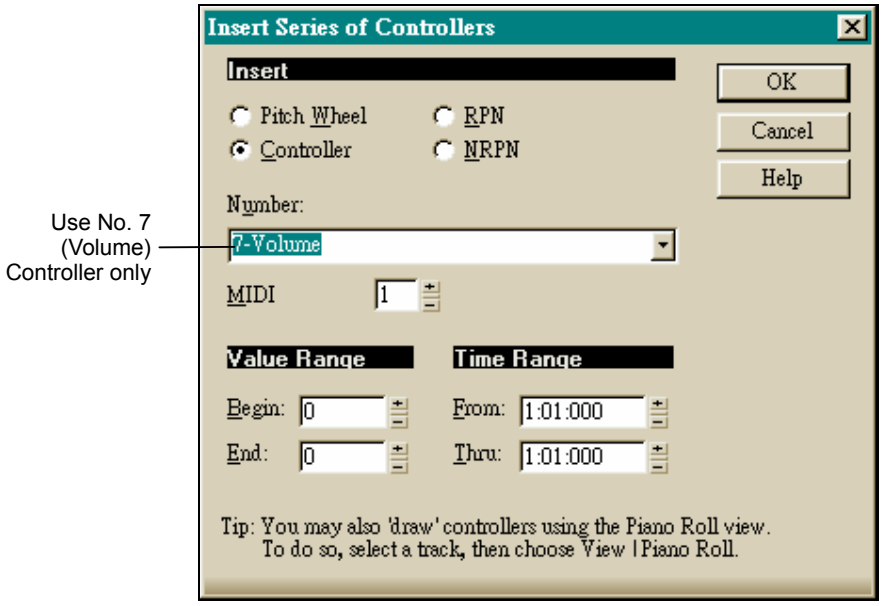
B.7 Meter

B.7.1 Limitations

- EM61000 only accepts one meter within a MIDI. If a song contains several meters, only the last meter will be recognized as the MIDI's meter. The last meter is then utilized as the applicable meter for the whole song.
- Furthermore, only a meter of n/4 is acceptable.

B.8 Controller

Of all the available MIDI Controllers, EM61000 can only interpret the No. 7-Volume. Avoid using the other controller numbers, as they will only be ignored.



MIDI Volume Controller Setup

Appendix C
Instrument Map (EM61.ist) File

C.1 Introduction

The Instrument Map (*.ist) file is applicable to EM61000 Series Integrated Development Environment (EM61IDE) Version 2.0 and supports all series of EM61000 chips (EM61001 ~ EM61500).

An Instrument Map (*.ist) file is automatically generated when you create a new MIDI project with the EM61IDE program. The file, which bears the same filenames as that of your project; is located under the project folder. You can always modify these files to suite the requirements of your project. An example of a modified Instrument Map file (TUTOR.IST) is shown below.

The file consisted of three major sections, namely**:**

1. [General] Section
2. [Instrument List] Section
3. [Template List] Section

The following shows the sample contents of each section. The subsequent Sections show their detailed function descriptions.

```
[General]
title=EM61000 Instrument File
version=2.0
$WORKDIR=D:\Work61\C61_0001\0927\TWF


[Instrument List]

tone group=1
tone group 0=T010 T024 T033 T046 T072 T073 T074 T091

percussion group=3
percussion group 0=P035
percussion group 1=P040
percussion group 2=P050
```

```
[T010]
name=Music Box
scale=A2-F#4 G4-F#5 G5-B6
A2-F#4=T010_0 %NORMAL %100
G4-F#5=T010_1 %NORMAL %100
G5-B6=T010_2 %NORMAL %100

[T024]
name=Acoustic Nylon Guitar
scale=C2-F#3 G3-F#4 G4-F#5 G5-F#7
C2-F#3=T024_0 %NORMAL %100
G3-F#4=T024_1 %NORMAL %100
G4-F#5=T024_2 %NORMAL %100
G5-F#7=T024_3 %NORMAL %100


[T033]
name=Electric Bass Fingered
scale=A2-F#4 G4-F#5 G5-B6
A2-F#4=T033_0 %NORMAL %100
G4-F#5=T033_1 %NORMAL %100
G5-B6=T033_2 %NORMAL %100

[T046]
name=Harp
scale=A2-F#4 G4-F#5 G5-B6
A2-F#4=T046_0 %NORMAL %100 DT046_0 %DETUNE %60
G4-F#5=T046_1 %NORMAL %100 DT046_1 %DETUNE %60
G5-B6=T046_2 %NORMAL %100 DT046_2 %DETUNE %60

[T072]
name=Piccolo
scale=A2-F#4 G4-F#5 G5-B6
A2-F#4=T072_0 %NORMAL %100 DT072_0 %DETUNE %85
G4-F#5=T072_1 %NORMAL %100 DT072_1 %DETUNE %85
G5-B6=T072_2 %NORMAL %100 DT072_2 %DETUNE %85

[T073]
name=Flute
scale=A2-F#4 G4-F#5 G5-B6
A2-F#4=T073_0 %NORMAL %100
G4-F#5=T073_1 %NORMAL %100
G5-B6=T073_2 %NORMAL %100

[T074]
name=Recorder
scale=A2-F#4 G4-F#5 G5-B6
A2-F#4=T074_0 %NORMAL %100
G4-F#5=T074_1 %NORMAL %100
G5-B6=T074_2 %NORMAL %100
```

```
[T091]
name=Chord Voice
scale=A2-B6
A2-B6=T091_0 %NORMAL %100


[P035]
name=Acoustic Bass Drum
template=P035f



[P040]
name=Electric Snare
template=P040f

[P050]
name=High Tom
template=P050f



[Template List]

t010_0=% DEFAULT 010_0.twf %0
t010_1=% DEFAULT 010_1.twf %0
t010_2=% DEFAULT 010_2.twf %0

t024_0=% USER $WORKDIR\024_C3.twf %0
t024_1=% USER $WORKDIR\024_C4.twf %0
t024_2=% USER $WORKDIR\024_C5.twf %0
t024_3=% USER $WORKDIR\024_C6.twf %0

t033_0=% DEFAULT 033_0.twf %0
t033_1=% DEFAULT 033_1.twf %0
t033_2=% DEFAULT 033_2.twf %0

t046_0=% USER $WORKDIR\046_C4.twf %0
DT046_0=% USER $WORKDIR\046_C4_D10.twf %0
t046_1=% USER $WORKDIR\046_C5.twf %0
DT046_1=% USER $WORKDIR\046_C5_D10.twf %0
t046_2=% USER $WORKDIR\046_C6.twf %0
DT046_2=% USER $WORKDIR\046_C6_D10.twf %0

t072_0=% USER $WORKDIR\072_C4.twf %4
DT072_0=% USER $WORKDIR\072_C4_D15.twf %4
t072_1=% USER $WORKDIR\072_C5.twf %4
DT072_1=% USER $WORKDIR\072_C5_D13.twf %4
t072_2=% USER $WORKDIR\072_C6.twf %4
DT072_2=% USER $WORKDIR\072_C6_D10.twf %4

t073_0=% USER $WORKDIR\073_C4.twf %4
t073_1=% USER $WORKDIR\073_C5.twf %4
t073_2=% USER $WORKDIR\073_C6.twf %4
```

```
t074_0=% USER $WORKDIR\074_C4.twf %4
t074_1=% USER $WORKDIR\074_C5.twf %4
t074_2=% USER $WORKDIR\074_C6.twf %4

T091_0=% USER $WORKDIR\091_G4.twf %0


p035f=% DEFAULT p035.twf %0
p040f=% DEFAULT P040.twf %0
p050f=% DEFAULT p050.twf %0
```

C.2 [General] Section

This section of the Instrument Map file defines the title and version information of the file and provides the path to your Tone-Waveform (*.twf) file.

C.2.1 [General] Example

```
[General]
title=EM61000 Instrument File
version=2.0
$WORKDIR=D:\Work61\C61_0001\0927\TWF
```

; The first two lines show the title and version of the file.
; The "$WORKDIR" (upper case) line is where you define the directory path of your twf file.

C.3 [Instrument List] Section

Instrument List is where you prescribe the instruments for your tone and percussion groups of sound. Architectures of its contents are explained below.

C.3.1 Tone/Percussion Instrument Group Field Contents

Field	Description
Tone group	Tone type instrument can be divided into several groups, or Tone group = n, where n is the tone group number.
Tone group n =	List the number used in the nth tone group. The values following the equal sign (=) states the instrument type (or MIDI patch number), prefixed with 'T" (ex: T010, T024. etc.). The numbers are separated by a space. Each instrument number represents one type of tone instrument and must be unique from others.
Percussion group	Percussion type instrument can be divided into several groups, or Percussion group = n, where n is the percussion group number.
Percussion group n =	List the number used in the nth tone group. The values following the equal sign (=) states the instrument number (or MIDI patch number), prefixed with "P" (ex: P035, P040, etc.). The numbers are separated by a space. Each instrument number represents one type of percussion instrument and must be unique from others.

C.3.1.1 Tone/Percussion Group Example

NOTE

*For formatting convenience, some portions of the example Instrument Map file shown below are displayed in two lines instead of the correct single line format as shown under Section A.1 (**Overview**). When writing your file, follow the formatting under Section A.1.*

```
[Instrument List]
tone group=1
tone group 0=T010 T024 T033 T046 T072
T073 T074 T091

percussion group=3
percussion group 0=P035
percussion group 1=P040
percussion group 2=P050
```

; Assign a required tone group for your project and collect the needed tone type instruments for such group i.e., Group 0 = T010 (Music Box); T024 (Acoustic Nylon Guitar); etc., or as you have defined for the project).
; Define the required percussion type instruments (in 3 groups) and assign each with the needed instrument; i.e., Group 0 = 035 (Acoustic Bass Drum); Group 1 = 040 (Electric Snare); and Group 2 = 050 (High Tom), or as you have defined for the project).

NOTE

1. *Tone group should only contain Tone type instruments and is prefixed with "T." Likewise, Percussion group should only contain Percussion type instruments and is prefixed with "P."*
2. *All instrument numbers enumerated in the tone or percussion group should have their respective instrument name and contents separately listed in the [Instrument List] section. Any omission will result to erroneous compilation of EM61IDE.*

C.3.2 Tone Instrument Pitch Scale Field Contents

Field	Description
[Tn]	Field name with tone instrument number (prefixed with "T") as defined by MIDI.
Name	The corresponding Instrument name for the above instrument number.
Scale	This field defines the full pitch range (gamut) in used and monitors the start and end points of each range (separated by a dash "-"). There can be more than one segment in a full pitch range. Each segment is separated from others by a space (ex: A2-F#4 G4-F#5 G5-B). Total number of segment in a pitch range cannot exceed 256.
pitch_scale	The individual segment of the above full pitch range (gamut). The information following the equal sign (=) states the corresponding properties of the particular segment (ex: A2-F#4=T010_0 %NORMAL %100).

C.3.2.1 Pitch Scale Format

```
pitch_scale = var(0) type weight [var(1) type weight]
```

Where:

`pitch_scale` is the name of the individual segment for the full pitch range (gamut) as defined in the "Scale" field.
*Example: **A2-F#4; G4-F#5, G5-B***

`var(0), var(1)…` are the instruments (template waveform) variables for each pitch scale (segment). These variables serves as indexes that link to the corresponding tone instruments properties defined in the Template List section.
*Example: **T046_0 / DT046_0; T046_1 / DT046_1, T046_2 / DT046_2***

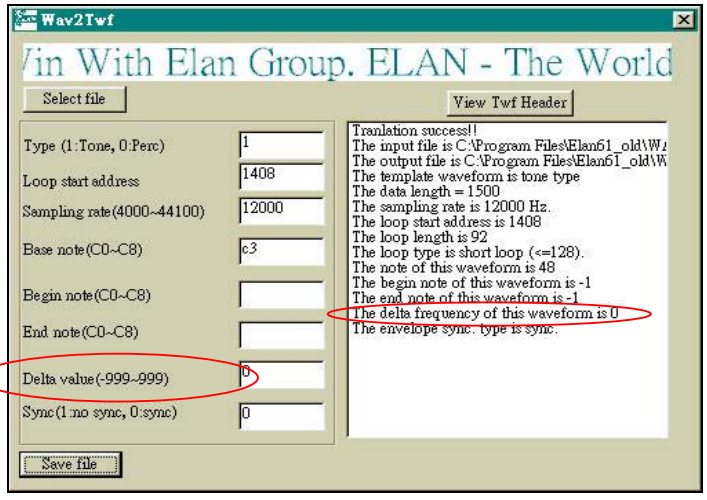
NOTE

For special effect and quality, no more than 2 template waveforms should be assigned for simultaneous playing per pitch scale (gamut).

`type` is either "%NORMAL" or "%DETUNE" (always in upper case). It specifies whether the waveform delta frequency compensation is required or not.

`type = %NORMAL:` when the tone instrument is played, waveform delta frequency is NORMAL (Delta value = 0, as set during conversion of WAV file into TWF file in the Wav2Twf tool. See figure below). Hence, no frequency compensation is required.

`type = %DETUNE:` frequency difference exists and frequency compensation is required when the tone instrument is played (Delta value ≠ 0, but is any number between –999 to 999 as defined in the Wav2Twf).



weight specifies the percentage of volume (based on the original sound) required when specified tone instrument is played.

Examples: **%0** zero percentage or none of the original volume is required, i.e., no sound (mute).

%60 requires 60% of the original volume when the specified tone instrument is played.

%100 requires 100% of the original volume when the specified tone instrument is played.

C.3.2.2 Pitch Scale Example

```
 [T010]
name=Music Box

scale=A2-F#4 G4-F#5 G5-B6
```

; This particular instrument number (***[T010]***) is the Music Box tone type instrument per MIDI.

; A2 ~ B6 is the defined full Scale range (gamut). It consisted of three pitch scales or segments, namely ***A2-F#4; G4-F#5; and G5-B6***.

NOTE

Number of segments in a full Scale range cannot exceed 256. Each segment should compose of a start pitch and an end pitch separated by a dash (-) A space separates the segments from others.

```
A2-F#4=T010_0 %NORMAL %100
G4-F#5=T010_1 %NORMAL %100
G5-B6=T010_2 %NORMAL %100
```

; Each of the 3 pitch scale (or segment) is assigned with a single tone instrument variable (***T010_0, T010_1***, and ***T010_2***) respectively. These variables are linked to tone instrument names stated in the [Template List] section where the complete properties of the instruments are defined.

Hence, A2-F#4 becomes

A2-F#4=T010_0 %NORMAL %100.

Where:

A2-F#4 is the defined pitch scale (segment).

T010_0 is the tone instrument variable to be sourced from [Template List].

%NORMAL is the waveform delta frequency value (0 = Normal or no frequency compensation required) specified when converting WAV file into TWF file with Wav2Twf tool (see figure in Page 6).

%100 is the required percentage (100%) of the original volume to be executed when the specified tone instrument is played.

NOTE

*%**NORMAL** or %**DETUNE** must be in upper case. The percent sign (%) has no function except to denote enclosure (open and close parenthesis).*

```
[T046]
name=Harp

scale=A2-F#4 G4-F#5 G5-B6
```

; This particular instrument name (***[T046]***) is the Harp tone type instrument according to MIDI.

; *A2 ~ B6* is the defined full Scale range (gamut). It consisted of three pitch scales or segments, namely ***A2-F#4***; ***G4-F#5***; and ***G5-B6***.

NOTE

Number of segments in a full Scale range cannot exceed 256. Each segment should compose of a start pitch and an end pitch separated by a dash (-) A space separates the segments from others.

```
A2-F#4=T046_0 %NORMAL %100 DT046_0
       %DETUNE %60
G4-F#5=T046_1 %NORMAL %100 DT046_1
       %DETUNE %60
G5-B6=T046_2 %NORMAL %100 DT046_2
      %DETUNE %60
```

; Each of the 3 pitch scale (or segment) is assigned with two tone instrument variables (***T046_0 / DT046_0***, ***T046_1 / DT046_1***, and ***T046_2 / DT046_2***) respectively. These variables link to tone instrument names stated in the ***[Template List]*** section where the complete properties of the instruments are defined.

Hence, ***A2-F#4*** becomes

A2-F#4=T046_0 %NORMAL %100 DT046 0 %DETUNE %60

Where**:**

A2-F#4 is the defined pitch scale (segment).

T046_0 / DT046 are the tone instrument variables to be sourced from ***[Template List]*** *section*.

%NORMAL is the waveform delta frequency value (0 = Normal or no frequency compensation required) specified when converting WAV file into TWF file with Wav2Twf tool (see figure in Page 6).

%DETUNE is the waveform delta frequency which value is –999 to 999, but NOT 0 (frequency compensation required when the tone instrument is played). This value is specified when converting WAV file into TWF file with Wav2Twf tool (see figure in Page 6).

%100 / %60 are the output volume setting in percentage. i.e., 100% for output without Delta_F change and 60% for output with Delta_F change.

NOTE

1. *For special effect and quality, no more than two tone instrument variables (template waveforms) should be assigned for simultaneous playing per pitch scale (gamut).*
2. ***%NORMAL** or **%DETUNE** must be in upper case. The percent sign (%) has no function except to denote enclosure (open and close parenthesis).*

```
[T091]
name=Chord Voice
scale=A2-B6
A2-B6=T091 0 %NORMAL %100
```

; This shows an example of a scale with a single pitch segment.

C.3.3 Percussion Instrument Field Contents

Field	Description
[Pn]	Field name with the percussion instrument number (prefixed with "P") as defined by MIDI
name	Instrument name
template	Template waveform variable name

C.3.3.1 Percussion Instrument Example

```
[P035]
name=Acoustic Bass Drum
template=P035f
```

; The percussion instrument (***[P035]***) is an Acoustic Bass Drum according to MIDI Channel 10. The ***template=P35f*** is linked to percussion instrument name stated in the ***[Template List]*** section where the complete properties of the instruments are defined.

NOTE

1. *Percussion instrument does not have scale range. It links to its properties in the **[Template List]** section through the "template" line.*
2. *Only one "template" line is allowed per percussion instrument.*

C.4 [Template List] Section

The Template List section defines the path of your twf file and the corresponding number of your tone instrument in the envelope file (*.env). These information are linked by the instrument parameters listed in the Instrument List section.

C.4.1 Field Contents

Field	Description
Var(n)	Assign a template waveform to this variable name and specify its file path and envelope index. The format is – `var(n) = location file_name envelope_index.`

C.4.2 Template Waveform Variable format

```
var = location file_name envelope_index
```

Where**:**

Var is the name defined by user for template waveform. Its length should not exceed over 32 characters and must be unique from others.

Location indicates the template waveform location in the default directory or otherwise.

location = %DEFAULT means template waveform is in the default directory (\instlib).

location = %USER means template waveform is not in the default directory. You must specify the full directory path.

File_name specifies the filename for the template waveform.

location = %DEFAULT need to specify the filename only.

location = %USER must specify the absolute path name or use **&WORKDIR**.

envelope_index specify the type of envelope for template waveform. The format is - **%n**, where *n* is any of the 32 types (0~32) envelope index available. Note that this applies to tone instruments only

C.4.2.1 Template Waveform Example

```
[Template List]

t010_0=% DEFAULT 010_0.twf %0

t072_0=% USER $WORKDIR\072_C4.twf %4
```

; The tone instrument (music box) is to be sourced from ***010_0.twf***, which is located in the DEFAULT folder ***\Inslib***.

; "***%0***" is the Envelop index number as defined in the Envelop (*.env) file.

; The tone instrument (Picolo) is to be sourced from ***072_c4.twf***, which is located in the USER defined ***TWF*** folder. Its path ***D:\Work61\C61_0001 \0927\TWF*** is defined in the **[General]** section described in Section 2.1.1 above.

; "***%4***" is the Envelop index number as defined in the Envelop (*.env) file.

```
p035f=% DEFAULT p035.twf %0
p040f=% DEFAULT p040.twf %0
p050f=% DEFAULT p050.twf %0
```

; The percussion instruments Acoustic Bass Drum (P035f), Electric Snare (P040f), and High Tom (P050f), are to be sourced from ***p035.twf,*** ***p040.twf,*** and ***p050.twf,*** respectively. These files are located in the DEFAULT folder ***\Inslib***.

; "***%0***" has no function in the Template List as unlike tone, percussion type instruments does not need percussion index. However, the value is needed to fill-in the program allocated slot. Otherwise, error will occur.

NOTE

1. ***DEFAULT, USER, & $WORKDIR*** *must be in upper case.*
2. *Envelope file index numbers are available from 0 to 31 only.*
3. *The parameter of each section is detailed in the *.env file of your Project folder.*
4. *The percent sign (%) has no function except to denote enclosure (open and close parenthesis).*

Appendix D
Envelope Data (EM61.env) File

D.1 Introduction

This Envelope (*.env) data file is applicable to EM61000 Series Integrated Development Environent (EM61IDE) Version 2.0. This tool supports all series of EM61000 chips (EM61001 ~ EM61500).

The Envelope (*.env) data file contains the tone waveform envelope parameters that are required and source d by twf files to play MIDI (in conjunction with the Instrument Map [*'ist] file) under EM61IDE. The full sound waveform envelope that is required to produce a complete sound feature to be played with your MIDI, are provided in the Envelope (*.env) data file.

A default Envelope (*.env) data file is automatically generated when you create a new MIDI project with the EM61IDE program. The file, which bears the same filenames as that of your project; is located under the project folder. You can always modify these files to suite the requirements of your project.

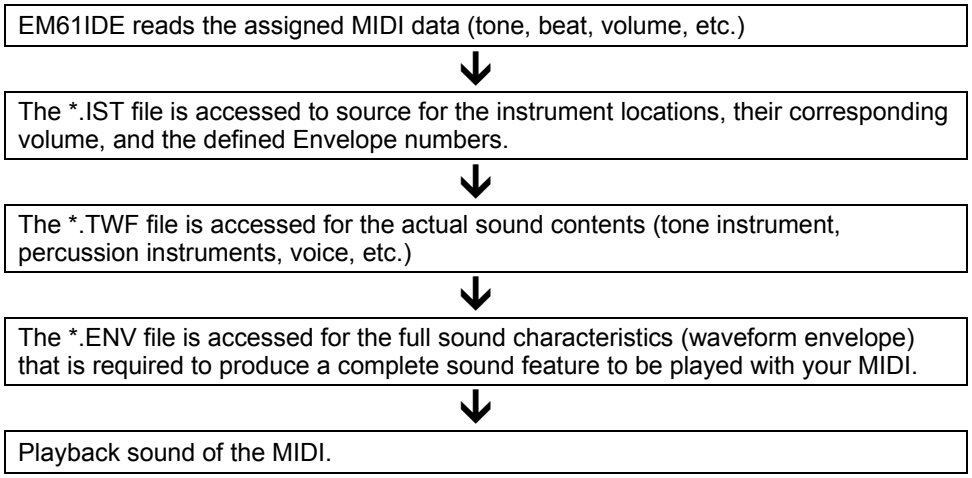
NOTE

*Existing *.env and *.ist files which were previously created and edited with EM61IDE Version 1.0, are supported by EM61IDE Version 2.0 under the following conditions:*

- *Both files must remain and used together as a pair, i.e., both files were created and edited using the EM61IDE Version 1.0.*
- *Neither of the files has been re-edited and overwritten under EM61IDE version 2.0.*

D.2 Envelop vs. Play MIDI

When your MIDI file is played through EM61IDE, the codes that defined the tones, together with their defined beats and volume variations; are read and played in the following sequence**:**


EM61IDE reads the assigned MIDI data (tone, beat, volume, etc.)
The *.IST file is accessed to source for the instrument locations, their corresponding volume, and the defined Envelope numbers.
The *.TWF file is accessed for the actual sound contents (tone instrument, percussion instruments, voice, etc.)
The *.ENV file is accessed for the full sound characteristics (waveform envelope) that is required to produce a complete sound feature to be played with your MIDI.
Playback sound of the MIDI.

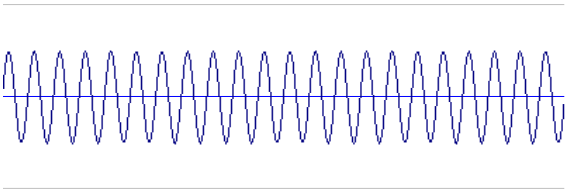
D.2.1 Summary

To summarize, MIDI playback sound may be summed up with the following formula**:**

Twf x **ist/100** x **env/127** x **midi/127 = speaker (output)**

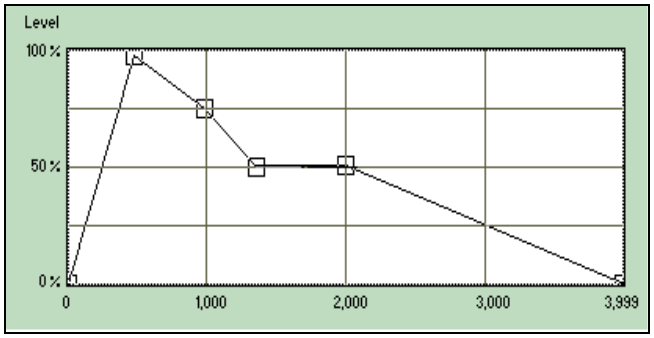
The full scale or maximum volume change of TWF, ENV, and MIDI is 127, while that of IST is 100. Any volume change to any of the files will affect the overall playback output

Figuratively speaking, the playback output may be illustrated as an integration of the basic waveform (TWF), sound volume change curve or envelope (ENV), and the MIDI file data as demonstrated in the following figures.



Waveform from TWF File

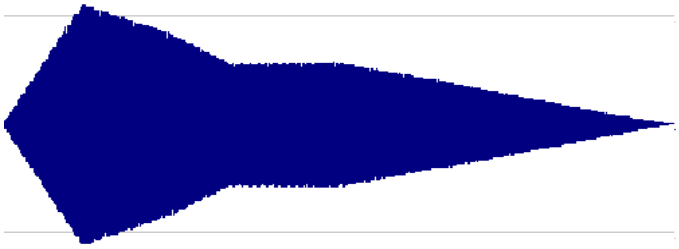
X (multiply)



Sound Curve Envelope from ENV File

X (multiply)

Sound Volume from MIDI File



Final Sound Output

D.4 Envelope Generator and Data Format

Envelope data of real sound could be achieved using a basic three-stage envelope generator, namely, **AD** (**Attack** + **Decay**), **Sustain**, and **Release**.

Attack-Decay (AD) depicts the time taken for the ascending amplitude to reach maximum (Attack) and the subsequent tapered decay rate (Decay). These rates of change in the envelope are shaped by samples. The duration for each sample is one tick.

Sustain is the next level of the envelope after Decay. It is further separated into two parts, Sustain Main and Sustain Vibration.

- **Sustain Main** where sustain level remains smooth but at slightly tapered angle
- **Sustain Vibration** where the sustain level is in a series of waveform segments (smooth repeating change of loudness).

Release is pure envelope decay stage, that is; the envelope value is decreased by M3 decrement at every tick.

D.4.1 Data Format

```
[Envelope n]                              ; n =  The assigned number (0~31)
                                          ;      of a particular Envelope.

AD group = m                              ; m =  The total number of AD groups
                                          ;      in a particular program.

AD group 0 = level  level  level          ; "level" specifies the volume levels

AD group m-2=level  level  level          ; with which the AD has to pass from
                                          ; the starting level to ending level
                                          ; (ranges from 0 to 127).

AD group m-1=level  … … last level -1     ; "-1" marks the final AD volume
                                          ; level and the end of the AD stage.
                                          ; It also marks the start of the
                                          ; Sustain stage.

Sustain main = m1 n1 Level                ; m1 = A constant value (0 ~ 127) or
                                           ;     rate with which the volume
                                           ;     levels decreases at each
                                           ;     defined tick value (n1) during
                                           ;     Sustain stage.
```

```
                                    ; n1 = The duration (number of tick at
                                    ;      48 Ticks per beat) required to
                                    ;      travel between the set constant
                                    ;      volume levels (m1).

                                    ; Level = The final decreasing volume
                                    ;      level with which the Sustain
                                    ;      stage ends and Release stage
                                    ;      starts.  This occurs when such
                                    ;      volume level is equaled or
                                    ;      negatively surpassed.

                                    ; EXAMPLE 1:
                                    ; Sustain main = 3 4 50
                                    ; Where m1=3, n1=4 ticks, & Level=50
                                    ; This means that the volume decreases
                                    ; at the rate of 3 levels per 4 ticks
                                    ; until the preset Final Volume level
                                    ; is equal or less than 50.

                                    ; EXAMPLE 2:
                                    ; Sustain main = 0 0 0
                                    ; Where m1=0, n1=0 ticks, & Level=0
                                    ; This means that the envelope has no
                                    ; Sustain stage and will hang at the
                                    ; last AD stage volume level.

                                    ; Note that the Sustain main is
                                       defined
                                    ; as = 0 0 0 when Sustain main is in
                                    ; horizontal (not tapered or
                                    ; decreasing) level, or in Vibration
                                    ; (waveform) state.

Sustain Vibration = m2 n2 Q         ; m2 = A constant value (0 ~ 127) or
                                    ;      volume level interval with
                                    ;      which the volume levels
                                    ;      increases or decreases at each
                                    ;      defined tick value (n2) during
                                    ;      the modulating Sustain Vibration;
                                    ;      (waveform) stage.

                                    ; n2 = the duration (number of ticks at
                                    ;      48 ticks/beat) required to
                                    ;      travel between the defined
                                    ;      volume level intervals (m2)in
                                    ;      both up & down modulation.
```

```
                                    ; Q =  the total number of volume
                                    ;      levels between the peak
                                    ;      and the lowest volume level in
                                    ;      a Vibration waveform.  (The
                                    ;      peak level is the last
                                    ;      volume level of the AD stage
                                    ;      where Sustain level starts.)

                                    ; EXAMPLE:
                                    ; AD last volume level = 70
                                    ; Sustain Vibration = 2 3 10
                                    ; This means the AD stage ends at
                                    ; volume level 70 and where Vibration
                                    ; modulation starts.  Volume further
                                    ; decreases from Level 70 to Level 60
                                    ; (70 - 10)at the rate of 2 levels per
                                    ; 3 ticks, and increases at the same
                                    ; rate from bottom Level 60 back to
                                    ; peak Level 70.
Release = X                         ; X =  A constant value (0 ~ 127) or
                                    ;      a rate with which the volume
                                    ;      levels decreases per tick
                                    ;      during Release (final decay)
                                    ;      stage.
```

D.5 Illustrative Examples and Demonstrations

D.5.1 Common Envelope

The following example shows a common envelope where its envelope Sustain Main ≠ 0 0 0, with playback in three-stages -

AD + Sustain main + Release

Example:

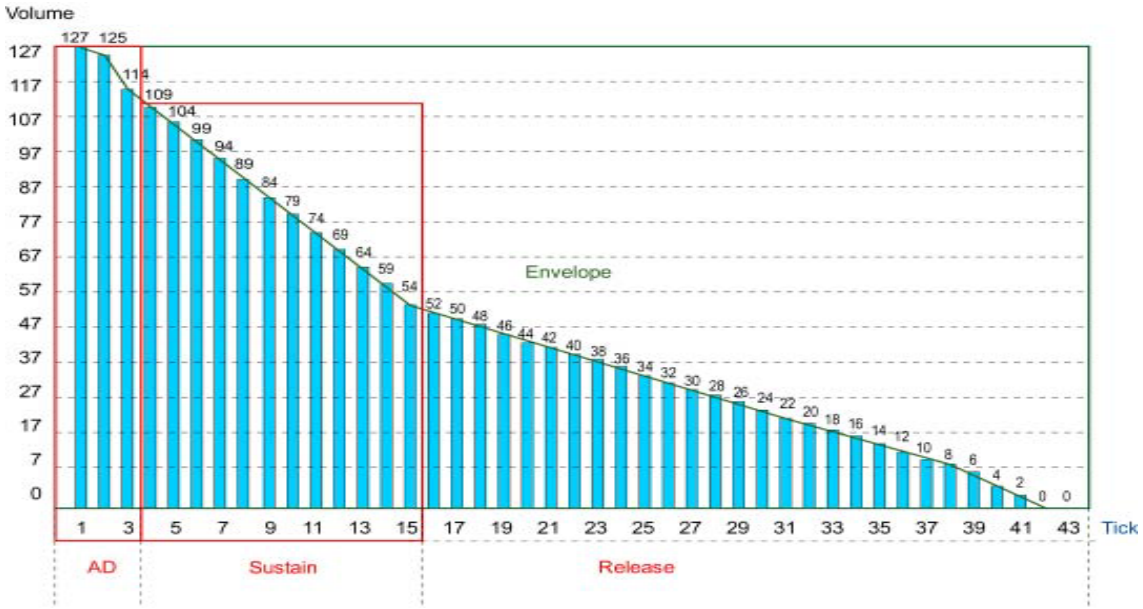
```
[Envelope 0]
AD group = 1
AD group 0 = 127  125  114  -1
Sustain main = 5  1  54

Sustain Vibration = 0 0 0

Release = 2
```

Meaning:

AD Stage starts at Volume Levels 127 (first level in AD Group 0), then at each tick, it decays (decreases) to Level 125, and ends at Level 114 where Sustain Stage begins.

Sustain Stage starts at Volume Level 114, then continues to decay at 5 levels interval at each tick (109 ➔ 104 ➔ 99 ➔……) until the volume level reaches Level 54 or below where Sustain Stage ends and Release Stage starts.

Release Stage starts at Volume Level 54, then continues further to decay at 2 levels interval per tick (52 ➔ 50 ➔ 48 ➔……) until the volume level reaches Level 0 where Release stage and the whole envelope ends.
The corresponding Sound Curve Envelope of the above envelope data is illustrated below**:**



D.5.2 AD Sustained Without Vibration Envelope

The following example shows a unique envelope where its Sustain Main = 0 0 0 and its Sustain Vibration is also = 0 0 0. The Release stage of this type of envelope will not function in accordance with the beat. The program will have the control when to release the Sustain stage before the end of a beat. Hence, the envelope will play in the following manner-

AD + Sustain AD last level + Programmed release

Example:

```
[Envelope 2]
AD group = 1
AD group 0 = 127  120  115  110  105  100  95  90  -1
Sustain main = 0 0 0
Sustain Vibration = 0 0 0
Release = 1
```

Meaning:

AD Stage starts at Volume Levels 127 (first level in AD Group 0), then at each tick, it decays (decreases) to Level 120, (120 ➔ 115 ➔ 110 ➔ 105 ➔……) and ends at Level 90 (last level in AD Group 0), where Sustain Stage begins.

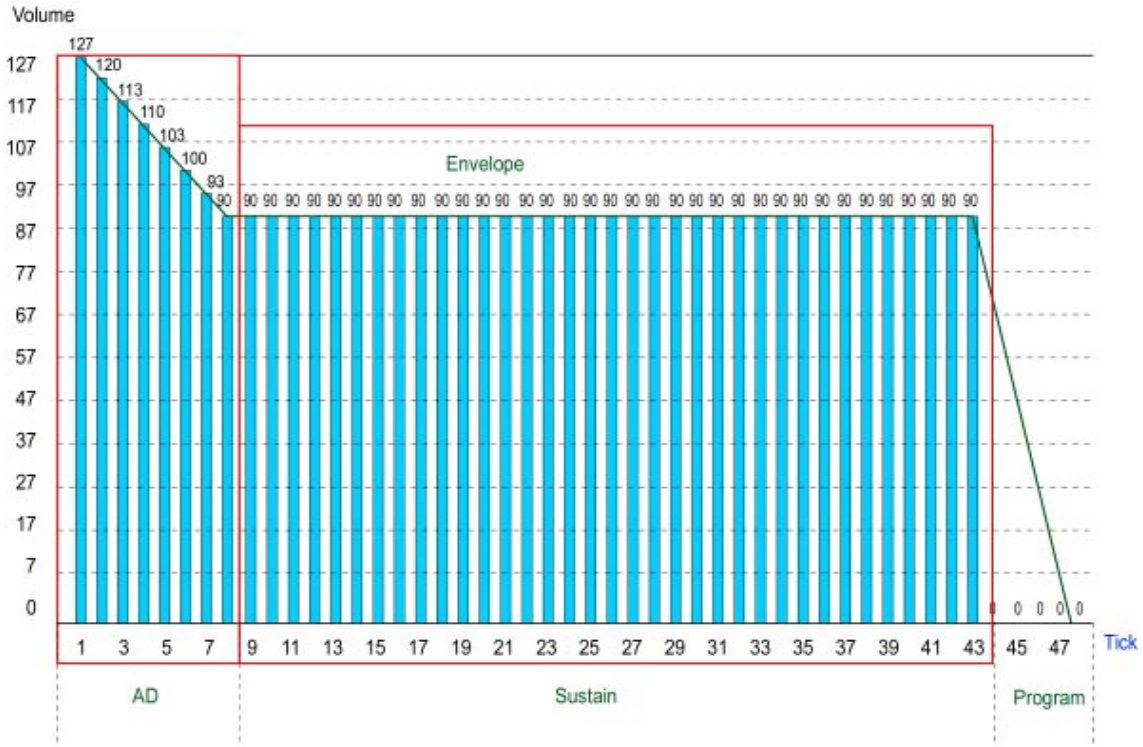
Sustain Stage starts at Volume Level 90 and the level is sustained horizontally (90 ➔ 90 ➔ 90 ➔……) until the program release instruction is received before the end of a beat.

Release Stage starts at the sustained Volume Level 90 and drops to Level 0 as it complies with the program instruction (➔ ➔ 0), and the whole envelope ends.

NOTE

The "Release = 1" in the data has no effect on this type for Envelope. However the line must exist and cannot be omitted. Otherwise, error will occur.

The resulting Sound Curve Envelope of the above envelope data is shown below**:**



D.5.3 AD Sustained with Vibration Envelope

The following example shows another unique envelope where its Sustain Main is equal to 0 0 0, but its Sustain Vibration is not. The Release stage of this type of envelope will not function in accordance with the beat. The program will have the control when to release the Sustain stage before the end of a beat. Hence, the envelope will play in the following manner-

AD + Sustain Vibration + Programmed release

Example:

```
[Envelope 3]
AD group = 1
AD group 0 = 127  120  115  110  105  100  95  90  -1
Sustain main = 0 0 0
Sustain Vibration = 4 1 8
Release = 1
```

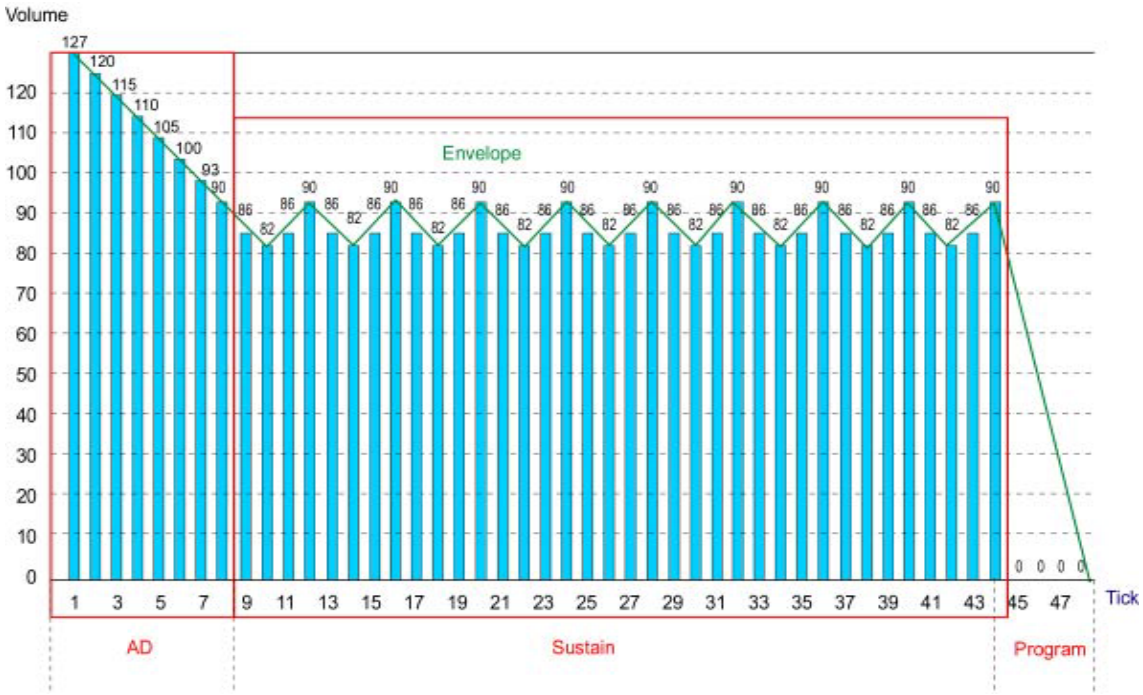
Meaning:

AD Stage starts at Volume Levels 127 (first level in AD Group 0), then at each tick, it decays (decreases) to Level 120, (120 ➔ 115 ➔ 110 ➔ 105 ➔……) and ends at Level 90 (last level in AD Group 0), where Sustain Stage begins.

Sustain Stage starts at Volume Level 90, then continues to decrease at 4 levels interval per tick until it reaches the bottom Volume Level 82 (90 – 8). Then increases at the same rate from bottom Level 82 back to peak Level 90 (90 ➔ 86 ➔ 82 ➔ 86 ➔ 90) where the Vibration (waveform) is repeated until the program release instruction is received before the end of a beat.

Release Stage also starts at the sustained Volume Level 90 and drops to Level 0 as it complies with the program instruction (➔ ➔ ➔0), and the whole envelope ends.

The resulting Sound Curve Envelope of the AD Sustained with Vibration using the above data, is shown below**:**


Volume
Envelope
Tick
AD
Sustain
Program